Managing users and groups

Grado en Informática 2024/2025
Departamento de Computación
Facultad de Informática
Universidad de Coruña

Antonio Yáñez Izquierdo

Contents I

Contents II

Users and groups

users

- A *user* is an entity in the system that can
  - Own files (and/or directories, devices . . .)
  - Create processes and execute programs
- A user is identified in the system by a number (*uid*)
- File permissions specify what a user can do to a spcific file
- There's a special user in the system (*uid=0*) that can access to all files, signal all processes and execute all system calls
- Every file in the system is **owned** by **ONE** user
- Process credentials indicate which user is behind the execution of that process

groups

- A *group* is a collection of users brought together for whatever reason
  - A *user* can belong to any number of groups (at least one)
  - A group can have any number of users
- A group is identified in the system by a number (*gid*)
- File permissions specify what users belonging to a group can do to a spcific file
- Every file in the system **belongs** only to **ONE** group, although the user owning it can be a member of several groups
- Of all the groups a user is member of, one is said to be his/her *primary* group

Managing user acounts

user database files

- as seen before, the system stores the information of users in plain text files
- these files are located in the `/etc` directory
- the format of these files is mostly standard across different unixes, although some of them are specific
- users can be added, deleted and have their accounts changed, by editing these files

list of user database files

- the files are:

`passwd`	this file defines the user acounts in the system. One line per user
`shadow`	pasword definition file, one line per user
`group`	group definition file, one line per group
`gshadow`	only on linux: the group password definition file
`master.passwd`	only on BSD systems. user definition file which includes passwords as well, protected from non root access

/etc/passwd file format I

- this is a plain text file where each line represents a user in the system
- the line is formed by fields separated by the **':'** character

```
username:x:UID:GID:user information:home-directory:login-shell
```

- **username**: this is the name of the user on the system
- **x**: on older systems the crypted form of the password (result of crypting a base block text with the password as key) was stored in this field
- **UID**: the number that represents the user in the system. this number mus be UNIQUE for each user. 0 always representes the system administrator, `root`. Multiple user accounts with the same UID are the same account from the system's point of view
- **GID**: user's login group number, also known as *primary group*

/etc/passwd file format II

- **user information**: these field, sometimes refered to as *gecos*, contains information about the user, such as its real name, department, phone . . .
- **home-directory**: the user's home directory. The user's shell is placed in that directory upon succesful login
- **login-shell**: the program started for the user when he/she logs in. Typically a command line interpreter or shell, like `/bin/sh` (Bourne shell), `/bin/csh` (C shell), `/bin/ksh` (Korn shell) `/bin/bash` (Bourne-Again shell) `tcsh` . . .

/etc/shadow file format I

- this file stores the password information
- it has one line per user in the system
- it is only readable by the system administrator
- the format of the line is

```
username:password:lastchg:min:max:warn:inactive:expire:reserved
```

- **username**: The user's login name
- **password**: The crypted form of the password, a special string (*,!,LK. . . depending on the system) indicates that the user can not login using the password. If left empty the user can log in without password
- **lastchg**: The date of the last password change, expressed as the number of days since Jan 1, 1970

/etc/shadow file format II

- **min**: The minimum number of days required between password changes. This field must be set to 0 or above to enable password aging
- **max** The maximum number of days the password is valid.
- **warn** The number of days before password expires that the user is warned
- **inactive** The number of days after a password has expired during which the password should still be accepted
- **expire** Date of expiration of the account, expressed as the number of days since Jan 1, 1970
- **reserved** This field is reserved for future use. On Solaris the four low order bits contain the failed login count

`/etc/master.passwd` file

- on BSD systems, the user information is stored in the `/etc/master.passwd` file
- it is protected from non root access
- for compatibility reasons, the `passwd` file is generated from it, using the `pwd_mkdb` command.
    - the `/etc/passwd` file gets all the information from `/etc/master.passwd` except the crypted form of the password (replaced by '$*$') and the pasword aging information
    - as in other systems, the `passwd` file is readable for every user on the system

/etc/master.passwd file format I

- it has one line per user in the system, in the following format
`username:pass:uid:gid:class:change:expire:gecos:home-dir:shell`
- **username**: this is the name of the user on the system
- **pass**: the crypted form of the password, a special string (*,!,LK. . . depending on the system) indicates that the user can not login using the password. If left empty the user can log in without password
- **uid**: the number that represents the user in the system.
- **gid**: user's login group number, also known as *primary group*

`/etc/master.passwd` file format I

- **class**: type of user, of several defined in the system
- **change**: password expiration time. Seconds since Jan 1, 1970
- **expire**: account expiration time. Seconds since Jan 1, 1970
- **home-dir**: the users home directory. the user is placed in that directory upon succesful login
- **shell**: the program started for the user when he/she logs in. Tipically a command line interpreter or shell

Creating a user

- to create a user the following tasks should be performed
  a. assign the user name, uid, password and primary group (creating it if necessary),
  b. create the corresponding entries in the user database files applying the system policies on password aging
  c. create the home directory of the user
  d. populate the user home directory with initialization files (typically found in `/etc/skel`) and create other necessary files (mail folder. . .)
  e. use `chown` and `chmod` to give the user ownership of its home directory and initialization files and give them the adecuate permissions

Creating a user

- tasks c), d) and e) can be easily achieved with simple straight-forward commands
- task a) usually involves careful thinking to decide on usernames, uids and, most important, passwords. System policies on password aging are supposed to be already decided when adding a user
- task b) must be done very carefully, as corrupting the user file databases could render the system unusable
    - some systems include the `vipw` command that invoques an editor to edit the `passwd` file, acting on a temporary copy and checking the file syntax before saving it

Changing shells

- to change the shell of a user the file `/etc/passwd` has to be changed
  - only the System Administrator, `root`, can modify this file
- user can change their shell with the command `chsh` (linux and BSD) or `passwd -e` (Solaris)
- for a user to be able to change his/her shell, the shell must be defined in `/etc/shells`

lmited accounts

- we can limit what we allow one user to do in the system
- if the user just uses the system to execute one program we can put that program in the /etc/password file as shell
  - for example, if a user just logs into a machine to read the mail, we can put the corresponding mail user agent in the /etc/passwd file as the shell for that user
- some shells also provide a restricted version (`rksh`, `rbash...`) with restricted funcionality that helps limiting the things a user can do in the system
- it is not a very good policy to put restricted shells in `/etc/shells`

Administrative tools for managing users

Tools for managing users

- there's several graphic tools to perform user administration tasks
- all of them are very complete and can perform almost any task
- unfortunately no one of the has become a *de-facto* standard and the tool we'll find will depend greatly on the O.S./graphic environment combination that we have
- on the other hand, text tools are much standarized with `useradd` being the most common

useradd

- it is the most standarized tool for adding users in one system
- available in almost any unix system (Solaris, BSD, linux. . .)
- can perform all the necesaary steps to create a new user
- two modes of operation
  - when invoked as `useradd -D` it stablishes the default options to be used when adding users to that system
  - invoked without the `-D` it adds a user to the system. Parameters supplied define the characteristics of the user account created
- Some recent BSD systems have substituted this comand for the *useradd* subcomand in the more general user administration command *pw*

other utilities

- **usermod** modifies a user's login information on the system
- **userdel** deletes a user account and, possibly, related files
- **adduser** (linux and BSD) more *user-friendly* utility to create users. Typically it's just a front end to `useradd`
- **rmuser** (BSD only) more *user-friendly* utility to delete users
- **pw** (some BSD systems only) mega command from which to invoke useradd, usermod, . . .

other considerations

- in addition to the `usermod` command, there are other utilities to control the password aging depending on the system
  - in Solaris we can control with the `passwd` command
  - linux has some options in the `passwd` command plus a specific `chage` command
- openBSD defines different classes of users, with, for example, different requirements as to password length or crypting algorhythm
- there are utilities to check for password strength, which can also be done through the use of PAM modules

Managing groups

`/etc/group` and `/etc/shadow` file format
the `nerwgrp` command
other group utilities

users and grups

- users are said to belong to one or more groups
- the group which GID appears in the line defining a user is called the user's *primary group*
- the user can also belong to other groups via the `/etc/group` file
- the command `groups` shows the groups a user belongs to

/etc/group file format

- this file is a plain text file with each line defining a group. The format of the line is

  `groupname:password:gid:user-list`
- **groupname** is the name of the group
- **password**: is the password of the group. This field is usually not used and has the character **'*'**. (linux has a separate `/etc/gshadow` file)
- **gid**: is the number identifying the group in the system
- **user-list**: comma separated list of the usernames of users belonging to that group (excluding the ones that have the group as their primary group)

/etc/gshadow file format

- this file is a plain text file defining the passwords of the group. It is specific to the linux operating system
- one line for each group. The format of the line is

  ```
  group_name:password:admin:user-list
  ```

  - **password** is the crypted form of the password
  - **admin** is a comma separated list of the usernames of the group administrators (users that can change the group password or members list)
  - **user-list**: comma separated list of the usernames of users belonging to that group

the `nerwgrp` command

- although one user can belong to several groups, the files he/she creates belong only to one group
- when a user logs in, the real and efective gids of the user shell are taken from `/etc/passwd` file
- all files and directories created get owned by that group
    - the group ownership of a file can be changed with `chgrp` to one of the groups the user belongs to
- the command `newgrp` allows one user to *login* to another group, by creating a new shell with real a effective gid to the ones of that group
    - if the user is a member of the group he/she wants to log in, `newgrp` succeeds
    - if the group has a password defined, the user can log to that group provided he/she knows the password

the `nerwgrp` command

- to assign a password to a group
  - a group administrator can do it with the `gpasswd` command
  - the root can do it copying and pasting the crypted form of a password from the `/etc/passwd` file
- BSD systems lack the `nerwgrp` command as the BSD group semantics specify that the group on a file is inherited from the parent directory: files created whithin a directory have the same group as the directory, regardless the gid of the creating process (on non BSD systems this behaviour is acomplished with the *setgid* bit on the direcory mode).

other group utilities

- as with the users, there are several graphic tools to perform group administration tasks. We wont't refer here to them, just list some of the standard commands to dealing with groups

groupadd	adds a group to the system
groupmod	modifies a group definition
groupdel	removes a group from the system
pw	with the adecuate subcomand can also manage the system's groups

BSD login classes

BSD login classes

- BSD systems define *login classes*
- login classes are defined in the file `/etc/login.conf`
- A login class is a user profile that imposes certain login capabilities and resource limitations to the users belonging to it
- The corresponding login class for a user is defined in the `/etc/master.passwd` database
- At least the *default* login class must exist. Additional *login classes* can be added to `/etc/login.conf`

User autentification with PAM

What is PAM?

- PAM stands for **P**luggable **A**uthentication **M**odules
- Provides a way of changing the authentication machanisms without changing the applications
- is a generalized API for authentication-related services
  - allows a system administrator to add new authentication methods simply by installing new PAM modules
  - allows a system administrator to modify authentication policies by editing configuration files
- available for Solaris, linux and BSD systems (openBSD uses BSD Authentication with is a different API)

What is PAM?

- lets consider the *login* program
  - once it reads the password, it compares its crypted form with the one in the `/etc/passwd` (or `/etc/shadow`) file
  - a change in the way the *crypted* password is stored or the way it is crypted would make necessary to recompile the login program
- Solution: PAM
  - PAM provides a library of functions that an application may use to request that a user be authenticated
  - changing anything in the authentication process would mean to change the PAM library, no the aplications: in fact most of the changes in the authentication process can be made just changing PAM configuration

Configuration of PAM

- There are different implementation of PAM and their configuration can differ slightly in
  - the location and format of the configuration file(s)
  - the location of the PAM library
  - list of available modules
- there is however a thing in common: lack of configuration means no authentication
  - Deleting PAM configuration file(s) locks you out of the system

PAM facilities

- we designate as facilities each of the tasks that PAM can deal with. These are
  - **authentication management (auth):** to determine whether the user is who he/she claims to be
  - **account management:** to handle non-authentication-related issues of account availability (for example, login at only certain hours or from certaing machines)
  - **session management:** to perform tasks associated with session set-up and tear-down, such as login accounting, stablishing resource limits. . .
  - **password management:** to change the authentication token associated with an account

PAM modules

- A PAM module is a self-contained piece of program code that implements the primitives in one or more facilities for one particular mechanism
- For a particular facility a module can be considered
  - **sufficient:** if this module grants access, access is granted, no more modules are checked
  - **requisite:** if this module denies access, access is denied, no more modules are checked
  - **required:** if this module denies access, access is denied, and the evalution continues with the following modules
  - **optional:** the result of this module will be used only if the result of no other modules is deterministic
  - **binding:** (FreeBSD only) success is sufficient; on failure all remaining modules are run, but the request will be denied.
  - **[new syntax]**: set of pairs of values. Not available in all implementations

PAM modules: new syntax

- apart from *sufficient*, requisite, required and optional, the control field in a pam configuration file can have the form

```
[value1=action1 value2=action2 ...valueN=actionN]
```

- where vauleJ can be one of the following: *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*, *conv_again*, *incomplete*, and *default*.
  - *default* stands for all values non explicitly listed.

PAM modules: new syntax (continuation)

- where actionJ can be one of the following

ignore	when used with a stack of modules, the module's return status will not contribute to the return code the application obtains.
bad	this action indicates that the return code should be thought of as indicative of the module failing. If this module is the first in the stack to fail, its status value will be used for that of the whole stack.
die	equivalent to bad with the side effect of terminating the module stack and PAM immediately returning to the application.

PAM modules: new syntax (continuation)

- ok — this tells PAM that the administrator thinks this return code should contribute directly to the return code of the full stack of modules. In other words, if the former state of the stack would lead to a return of PAM_SUCCESS, the module's return code will override this value. Note, if the former state of the stack holds some value that is indicative of a modules failure, this 'ok' value will not be used to override that value.

  done — equivalent to ok with the side effect of terminating the module stack and PAM immediately returning to the application.

  reset — clear all memory of the state of the module stack and start again with the next stacked module.

PAM modules:new syntax (continuation)

- In fact, the control words *sufficient*, *requisite*, *required* and *optional* can be expressed in the new syntax, as follows
  - **[required]** [success=ok new_authtok_reqd=ok ignore=ignore default=bad]
  - **[requisite]** [success=ok new_authtok_reqd=ok ignore=ignore default=die]
  - **[sufficient]** [success=done new_authtok_reqd=done default=ignore]
  - **[optional]** [success=ok new_authtok_reqd=ok default=ignore]

A little example

- Consider the following example related to the *su* service

```
auth    sufficient    pam_rootok.so
auth    required      pam_wheel.so
auth    required      pam_unix.so
```

- inferring what the modules do from their name, this configuration of the *su* service states that
    - the access would be granted directly for the root user
    - other users would have to both belong to the wheel group and enter the correct password

PAM files

- the module files usally are located
  - solaris and BSD systems: `/usr/lib/security`
  - linux systems: `/lib/security`, `/lib64/security` or `/lib/x86_64-linux-gnu/security`
- On Solaris the configuration file used to be `/etc/pam.conf`. From Solaris 11 on, we have an `/etc/pam.d` directory
- On linux and BSD the configuration files reside in the `/etc/pam.d` directory. In absence of `/etc/pam.d`, `/etc/pam.conf` is checked for
- Alternatively on BSD systems `/usr/local/etc` and `/usr/local/etc/pam.d` are checked
  - there is one file per service to be configured
  - the file is named after the service it configures

/etc/pam.conf file format

- plain text file.
- lines starting with # are comments
- each line has the format

```
service_name     facility  control_flag   module  options
```

`/etc/pam.conf` file format

- `service_name` is the name of the service to be configured, for example *sshd*, *telnetd*, *su* . . .
- `facility` is one of: *auth*, *session*, *account*, *password*
- `control_flag` states how the module affects the facility for that service, and can be: *sufficient*, *requisite*, *required*, *optional*
- `module` is the name of the modules (older versions of PAM used the complete path to the modules files)
- `options` are the parameters passed to the module in case the module accepts (or requires) options to be passed to it.

/etc/pam.conf example

- The previous example would look like this in an `/etc/pam.conf` file

```
su        auth      sufficient      pam_rootok.so
su        auth      required        pam_wheel.so
su        auth      required        pam_unix.so
```

- services not explicitly defined use the modules defined in the *“other“* section

/etc/pam.d

- should this directory exist, the `/etc/pam.conf` is not read
- there is a plain text file in the directory `/etc/pam.d` for each service to be configured
- each line in these files
  - is considered a comment if it starts with #
  - has the format

    ```
    facility   control_flag   module    options
    ```
  - the following syntax (no universally uderstood)causes to include another configuration file in the present service (useful to have common policies for different services)

    ```
    @include other_file_in_the_pam.d_directory
    ```

PAM modules

- the list of PAM modules dependes on the PAM implementation
  - each pair OS/PAM implementation may have a different set of modules
- info on the modules can be obtained with the man page
- there are, however, modules that are common to almost every implementation
- some modules with the same name behave differently on different implementations

some common PAM modules

- `pam_deny` locks out PAM modules
- `pam_getenv` returns the value for a PAM environment name
- `pam_rhosts` `pam_rhosts_auth` the rhosts PAM module
- `pam_unix` `pam_unix_auth` PAM authentication module for UNIX
- `pam_winbind` PAM module for winbind

basic linux PAM modules

- pam_permit always grants access
- pam_deny locks out PAM modules.
- pam_access delivers log-daemon-style login access control using login/domain names depending on pre-defined rules in /etc/security/access.conf.
- pam_cracklib checks the passwords against the password rules.
- pam_env sets/unsets environment variables from /etc/security/pam_env_conf.
- pam_debug debugs PAM.

basic linux PAM modules

- pam_echo prints messages.
- pam_exec executes an external command.
- pam_ftp is the module for anonymous access.
- pam_localuser requires the user to be listed in /etc/passwd.
- pam_unix provides traditional password authentication from /etc/passwd.

list of linux PAM modules I

```
pam_access (8)        - PAM module for logdaemon style login access control
pam_ck_connector (8) - Register session with ConsoleKit
pam_debug (8)         - PAM module to debug the PAM stack
pam_deny (8)          - The locking-out PAM module
pam_echo (8)          - PAM module for printing text messages
pam_env (8)           - PAM module to set/unset environment variables
pam_exec (8)          - PAM module which calls an external command
pam_filter (8)        - PAM filter module
pam_ftp (8)           - PAM module for anonymous access module
pam_getenv (8)        - get environment variables from /etc/environment
pam_group (8)         - PAM module for group access
pam_issue (8)         - PAM module to add issue file to user prompt
pam_keyinit (8)       - Kernel session keyring initialiser module
  ºQ ºQ234Y13 ºpam_lastlog (8)      - PAM module to display date of last login
pam_limits (8)        - PAM module to limit resources
pam_listfile (8)      - deny or allow services based on an arbitrary file
pam_localuser (8)     - require users to be listed in /etc/passwd
pam_loginuid (8)      - Record user's login uid to the process attribute
pam_mail (8)          - Inform about available mail
pam_mkhomedir (8)     - PAM module to create users home directory
```

list of linux PAM modules II

```
pam_motd (8)           - Display the motd file
pam_namespace (8)      - PAM module for configuring namespace for a session
pam_nologin (8)        - Prevent non-root users from login
pam_permit (8)         - The promiscuous module
pam_pwhistory (8)      - PAM module to remember last passwords
pam_rhosts (8)         - The rhosts PAM module
pam_rootok (8)         - Gain only root access
pam_securetty (8)      - Limit root login to special devices
pam_selinux (8)        - PAM module to set the default security context
pam_sepermit (8)       - PAM module to allow/deny login depending on SELinux en..
pam_shells (8)         - PAM module to check for valid login shell
pam_tally (8)          - The login counter (tallying) module
pam_time (8)           - PAM module for time control access
pam_timestamp (8)      - Authenticate using cached successful authentication at..
pam_umask (8)          - PAM module to set the file mode creation mask
pam_unix (8)           - Module for traditional password authentication
pam_userdb (8)         - PAM module to authenticate against a db database
pam_warn (8)           - PAM module which logs all PAM items if called
pam_wheel (8)          - Only permit root access to members of group wheel
pam_winbind (8)        - PAM module for Winbind
pam_xauth (8)          - PAM module to forward xauth keys between users
```

list of Solaris PAM modules I

```
pam_authtok_check (5)   - authentication and password management module
pam_authtok_get pam_authtok_get (5) - authentication and password management mo
pam_authtok_store (5)   - password management module
pam_deny (5)      - PAM authentication, account, session and password management
pam_allow (5)     - PAM module to allow operations
pam_dhkeys (5)    - authentication Diffie-Hellman keys management module
pam_dial_auth (5)   - authentication management PAM module for dialups
pam_krb5 (5)      - authentication, account, session, and password management PAM
pam_krb5_migrate (5)    - authentication PAM module for the KerberosV5 auto-mig
pam_ldap (5)      - authentication and account management PAM module for LDAP
pam_list (5)      - PAM account management module for UNIX
pam_passwd_auth pam_passwd_auth (5) - authentication module for password
pam_projects (5)    - account management PAM module for projects
pam_rhosts_auth pam_rhosts_auth (5) - authentication management PAM module usin
pam_roles (5)     - Solaris Roles account management module
pam_smartcard (5)   - PAM authentication module for Smart Card
pam_tsol_account (5)    - PAM account management module for Trusted Extensions
pam_unix_account (5)    - PAM account management module for UNIX
pam_unix_auth (5)   - PAM authentication module for UNIX
```

list of Solaris PAM modules II

```
pam_unix_cred (5)   - PAM user credential authentication module for UNIX
pam_unix_session (5)    - session management PAM module for UNIX
pam_winbind (1m)    - PAM module for Winbind
```

list of FreeBSD PAM modules I

```
pam_chroot(8)            - Chroot PAM module
pam_deny(8)              - Deny PAM module
pam_echo(8)              - Echo PAM module
pam_exec(8)              - Exec PAM module
pam_ftpusers(8)          - ftpusers PAM module
pam_group(8)             - Group PAM module
pam_guest(8)             - Guest PAM module
pam_krb5(8)              - Kerberos 5 PAM module
pam_ksu(8)               - Kerberos 5 SU PAM module
pam_lastlog(8)           - login accounting PAM module
pam_login_access(8)      - login.access PAM module
pam_nologin(8)           - NoLogin PAM module
pam_opie(8)              - OPIE PAM module
pam_opieaccess(8)        - OPIEAccess PAM module
pam_passwdqc(8)          - Password quality-control PAM module
pam_permit(8)            - Promiscuous PAM module
pam_radius(8)            - RADIUS authentication PAM module
pam_rhosts(8)            - Rhosts PAM module
pam_rootok(8)            - RootOK PAM module
pam_securetty(8)         - SecureTTY PAM module
```

list of FreeBSD PAM modules II

```
pam_self(8)            - Self PAM module
pam_ssh(8)             - authentication and session management with SSH priva
pam_tacplus(8)         - TACACS+ authentication PAM module
pam_unix(8)            - UNIX PAM module
```

list of NetBSD PAM modules I

```
pam_radius (8) RADIUS PAM module
pam_exec (8) Exec PAM module
pam_echo (8) Echo PAM module
pam_guest (8) Guest PAM module
pam_permit (8) Promiscuous PAM module
pam_ftpusers (8) ftpusers PAM module
pam_rootok (8) RootOK PAM module
pam_rhosts (8) rhosts PAM module
pam_self (8) Self PAM module
pam_securetty (8) SecureTTY PAM module
pam_deny (8) Deny PAM module
pam_nologin (8) NoLogin PAM module
pam_group (8) Group PAM module
pam_chroot (8) Chroot PAM module
pam_unix (8) UNIX PAM module
pam_afslog (8) AFS credentials PAM module
pam_skey (8) S/Key PAM module
pam_lastlog (8) login accounting PAM module
pam_krb5 (8) Kerberos 5 PAM module
pam_ksu (8) Kerberos 5 SU PAM module
pam_login_access (8) login.access PAM module
```

Solaris' roles

Conventional UNIX systems

- In traditional UNIX systems, the *privileged* user `root` has all the rights
  - Access to all files
  - Execute all system calls
  - Signal all procesess
  - ...
- A process running as `root` has complete power on the system
- A user that gets access to the system as `root` has complete power on the system

Least privileges principle

- *"least privileges"* is a concept in security. Give someone only the privileges needed to only do the task they are assigned
- For example: someone who has to create a user account does not need the privileges necessary to shut the system down, or to change the network configuration or ...
  - Having to be root to create a user account does not meet the principle of *least privileges*
- The Solaris O.S. addresses this problem with the implementation of *roles*
- Role Based Access Control is designed around the principle of *least privileges*

Role concepts

- Solaris *Role Based Access Control* defines the following concepts: *privileges*, *authorizations*, *rights profiles* and *roles*

privilege	A discrete right that can be granted to a command, a user or a role
authorization	A permission that enables a user or role to perform a class of actions that require additional rights
rights profile	A collection of security attributes that can be assigned to a role or to a user. A rights profile can include authorizations, directly assigned privileges, commands with security attributes, and other rights profiles. Profiles that are within another profile are called supplementary rights profiles
role	A special identity for running privileged applications. The special identity can be assumed by assigned users only

Example of RBAC Element Relationships

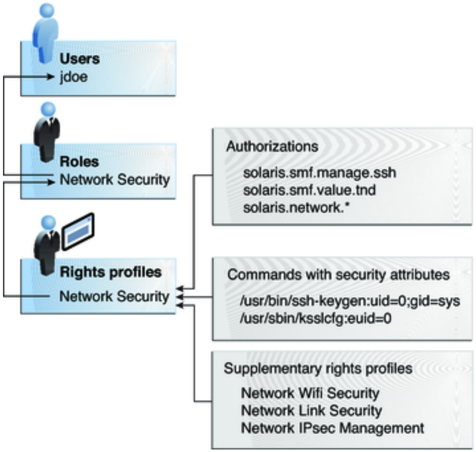

Figure: http://docs.oracle.com/cd/E23824_01/html/821-1456/figures/rbac-ex.png

Role concepts

- Solaris has some (few) predefined roles. We assign roles to users so they can assume the role temporarily to perform certain tasks
- We can also create new roles from the rights profiles predefined in the system (these roles will be typically named the same as the *rights profiles* they derive from) and assign them to users
- We can create new rights profiles from existing rights profiles or privileges, then create new roles from them, which will be assigned to users

Role implementation

- Roles are implemented like users (although they use pf*sh shells and they cannot login in the system directly)
- Users can asume a role via de su (*substitute user*) command
- The command `profiles` creates, modifies or examines right profiles
- The commands `roleadd`, `roledel`, and `rolemod` add, delete and modify roles in the system

Role implementation

- The command `roles` lists the roles assigned to users
- Users can be assigned roles using the `usermod` command
- Implementation resides in the following files in the `/etc/security` directory

`auth_attr`	authorization description database
`prof_attr`	profile description database
`exec_attr`	execition profiles database

Roles and root user

- In Solaris 11, if a user is created during installation `root` becomes a role
- `root` being a role implies that
  - `root` cannot login directly
  - For any user knowing the root password to become root, he/she has to have the root role asigned to him/her (via the roleadd command)
- The user created during the O.S. installation has the `root` role assigned to him/her

Example 1:Creating a role from existing rights profiles

- We want users *manuel* and *jose* to be able to manage users and software packages in our system
- The command `profiles -a` lists all the profiles and we find that there is a predefined rights profile *'User Management'* for managing users and a profile *'Software Installation'* for managing software packages
- We create the role `upadmin` with the pfbash shell and those rights profiles

```
#roleadd -c "Administrador usuarios y software" -s /usr/bin/pfbash -m \
           -K profiles="Software Installation,User Management" upadmin
```

Example 1:Creating a role from existing rights profiles

- We define a password for the role

```
#passwd upadmin
```

- We communicate the password to users `manuel` and `jose` and assign the role to them

```
#usermod -R +upadmin manuel
#usermod -R +upadmin jose
```

- users *manuel* and *jose* can perform user and software management tasks by assuming de *upadmin* role

```
manuel@maquina:~$ su upadmin
```

Example 2:Creating a role from non existing rights profiles

- Lets assume whe want an unplivileged user (*antonio*) to be able to shutdown the system
- First we create the profile to shutdown the system. We add the following line to the */etc/security/prof_attr* file

```
Shutdown:::Perfil para apagar:help=shutdown.html
```

  - Note that we didn't add any specific authorisation in the profile
- Next we add the capability to execute the shutdown command as root to this profile. We do this by adding the following line to the */etc/security/exec_attr*

```
Shutdown:solaris:cmd:::/usr/sbin/shutdown:uid=0
```

Example 2:Creating a role from non existing rights profiles

- We now create de role *apagar*

  ```
  #roleadd -c "Apagador del sistema" -m apagar
  ```

- Assign the role the profile *Shutdown* and the password to the role *apagar*

  ```
  # rolemod -P Shutdown apagar
  # passwd apagar
  ```

- Finaly we assign the role *apagar* to user *antonio*

  ```
  # usermod -R +apagar antonio
  ```

sudo and *sudoers*

sudo and *sudoers*

- one problem with the *su* command ist that it gives you access to the root account in an **all or nothing** fashion
- if you become *root*, you have **ALL** the privileges of the root account.
- maybe we'd like to just allow some users to perform certain administration task. The *sudo* command allows a user, after authenticating as his/herself, execute some command with administrator privileges, provided the *sudores* file allows him to. Example

```
user@somemachine $ sudo shutdown -h now
```

sudo

- sudo (and the *sudores file*) are avilable in linux distros and the solaris 11 O.S.
- the general syntax of the sudo command is

  ```
  sudo targetuser command
  ```

  so, provided that the user issuing the command is authorized to run *command* as *targetuser* in the *sudoers* file
  - user will be prompted for HIS/HER password (not targetuser's)
  - *command* will be executed with targetuser's credentials

sudoers file

- usually located at `/etc/sudoers`. Configuration can be appended at `/etc/sudoers.d`
- should not be edited directly but with the command *visudo*
  - *visudo* checks the syntax is correct before saving the file. In case there's an error in the syntax the sudo command will be disabled, so *visudo* prevents us from accidentally disabling *sudo*
- this file is formed by a series of lines in the form

```
user-spec   host-spec = (runasuser-spec) command-spec
```

- this sample line allows user antonio to run the command shutdown as root in host abyecto

```
  antonio  abyecto=(root) shutdown
```

- as the *sudoers* file is checked locally, the host-spec only makes sense when we have a common *sudoers* file for several machines

sudoers file

- the `user-spec` can be an username, an #userid, a %groupname or a %#groupid, an *alias* or a list of those elements separated by comma (`,`)
- the `host-spec` can be a hostname, a qualified hostname, a host address, a network address, an *alias* or a list of those elements separated by comma (`,`)
- the `runasuser-spec` can be an username, an #userid, a %groupname or a %#groupid, an *alias* or a list of those elements separated by comma (`,`)
- the `command-spec` can be a command name, an alias or a list of those elements separated by comma (`,`)
- any of those `*-spec` can be '**ALL**', specifying any user, host, or command

sudoers file

- aliases can be defined with
  TypeOfAlias ALIASNAME = list of members in that alias
  where TypeOfAlias can be *User_Alias, Runas_Alias, Host_Alias* and *Cmnd_Alias*
- The following example shows how to allow users *pepe, pepa and user2* to execute any of the commands that can power down the machine *rutercillo*

```
User_Alias  DOWNDOERS =  pepe, pepa, user2
Cmnd_Alias  POWERDOWN = /sbin/shutdown, /sbin/halt, /sbin/reboot, /sbin/restart

DOWNDOERS  rutercillo=(root) POWERDOWN
```