sicurezza dei sistemi unix/linux

sommario

- controllo di accesso in generale e nel filesystem
- autenticazione e login
- logging

unix – controllo di accesso

controllo di accesso in unix

- i processi accedono alle risorse (file e altro) tramite il kernel
  - mediante system call
- il controllo di accesso viene eseguito dal kernel
- in unix il “soggetto” del controllo di accesso è un processo ed è identificato dalle sue **credenziali**

credenziali di un processo unix

a ciascun processo unix è associato…

- UID: id utente “reale”, usato per tracciare chi ha lanciato il processo (es. dal comando ps)
- **EUID: “effective” UID, usato per access control**

- GID: gruppo principale “reale”, per tracciamento
- **EGID: gruppo principale “effettivo”, usato nell’access control**
  - ...e come gruppo quando si crea un file
- **supplementary groups: usati nell’access control**

real vs. effective

per quasi tutti i processi UID=EUID e GID=EGID

ma non sempre...

- vedi suid/sgid più avanti

processi privilegiati e non

- processi privilegiati
  - **EUID=0**, cioè processi che girano “come root”
  - il kernel non limita tali processi
- processi non privilegiati
  - **EUID≠0**
  - le operazioni ammesse dal kernel sono alcune

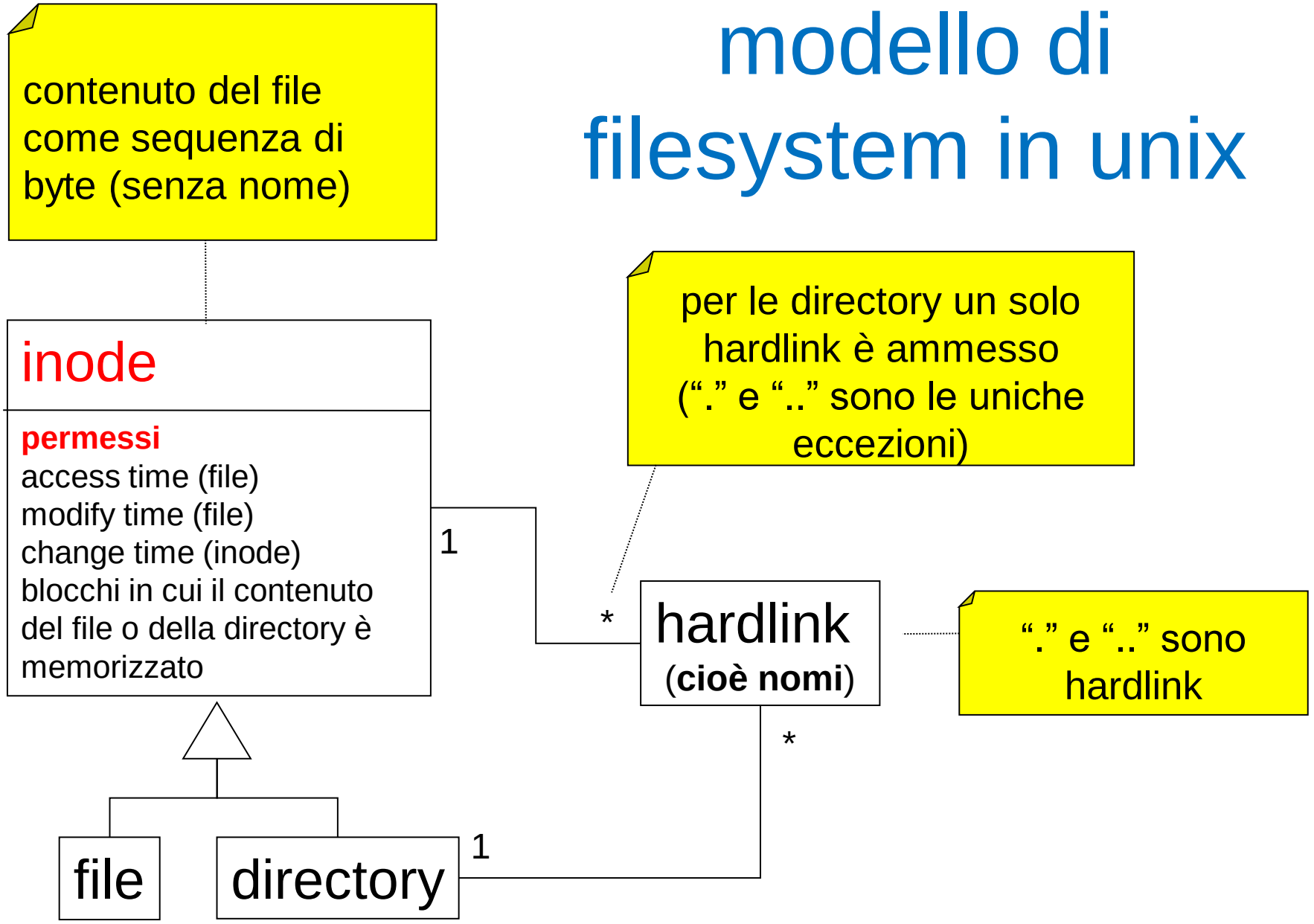
diritti dei processi **non** privilegiati

- filesystem
  - **creare e cancellare file (links) in directory in accordo con i permessi configurati per la directory**
  - **aprire** file in accordo con i permessi configurati per il file
    - per qualsiasi operazione su un file gia' aperto il controllo di accesso è solo contro la “modalità di apertura” e non contro i permessi del file
  - **cambiare permessi** dei “propri” file/directory
  - **cambiare proprietario** dei “propri” file/directory
- rete
  - usare socket regolari (tcp, udp, unix, **no raw socket** i.e. no ping)
  - binding di port porte >=1024
- processi
  - kill o ptrace su processi dello stesso utente
    - “kill” comprende tutti i segnali come stop, cont, ecc.
  - possibili altri vincoli, es. ptrace solo sui figli (YAMA LSM)

diritti dei processi privilegiati

- un processo *privilegiato* può praticamente tutto, ecco alcuni esempi…
- **filesystem**
  - nessuna limitazione
  - **cambiare permessi** e **proprietario** di tutti i file
- rete
  - usare qualisiasi tipo di socket, anche raw
  - binding di well-known ports <1024
  - amministrazione: interfacce, tabella di routing
  - rete in modalità promiscua e generazione di pacchetti qualsasi
- processi
  - inviare qualsiasi segnale e ptrace su qualsiasi processo
  - renice
  - cambio delle credenziali (necessario per login degli utenti)
- varie
  - (u)mount, quota, swap
  - reboot, shutdown, chroot, kernel modules, system clock

modello di filesystem in unix

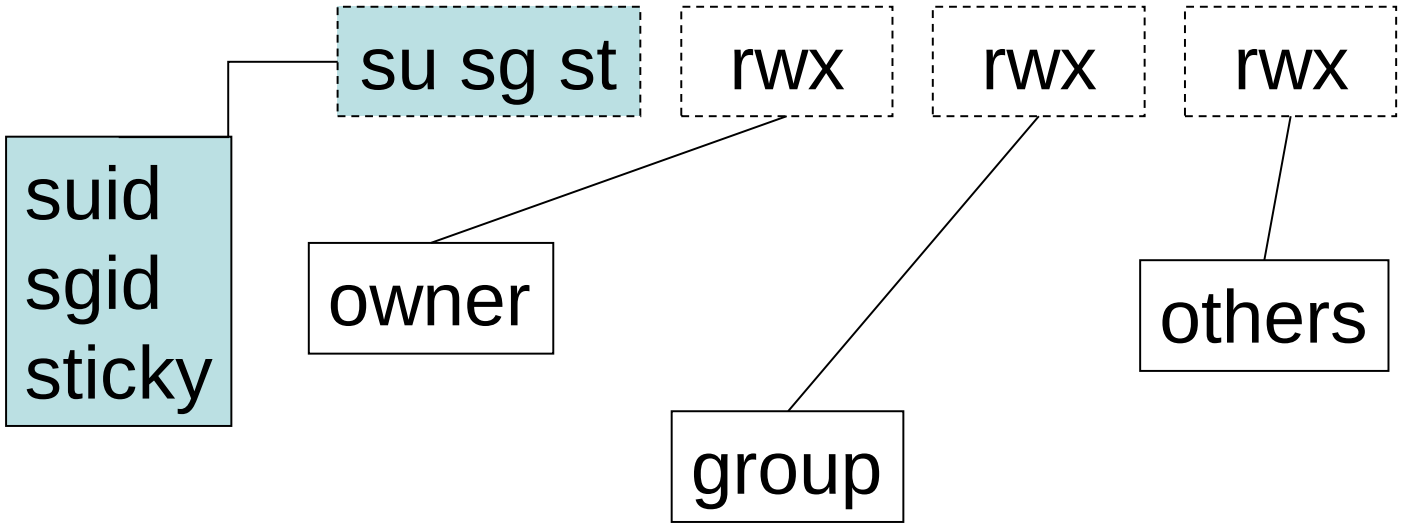


system calls per hardlink: link e unlink

- link(srg,dest): creazione di un hardlink dest che punta allo stesso inode di srg
  - da shell si usa il comando “ln srg dest”
- unlink(x): rimozione di un hardlink x
  - un inode viene rimosso quando non ha più alcun hardlink che lo punta
  - da shell si usa il comando “rm x”

filesystem unix: permessi

- permessi su inode per file
  - read (r), write (w), execute (x)
- permessi su inode per directory
  - read (r), write (w), **search** (x)
- a ciascun inode sono associati tre gruppi (triplette) di permessi chiamati *owner, group, others*
  - più altri



filesystem unix: chmod

- chmod <*chi*> +/-/= <*cosa*> pippo.txt
  - chi: u=user, g=gruppo, o=other
  - chmod ug+rw pippo.txt (set read e write per user e group
  - chmod o-x pippo.txt (unset execution per gli altri)
  - chmod ug=r (set r per user e group e unset il resto)
- sinstassi “ottale” ancora usata
  - “chmod 660 pippo.txt” configura rw- rw- ---

algoritmo per access control

- input: le syscall per aprire file (open) e directory (opendir) specificano l'oggetto da aprire come **pathname**
  - ciascun "name" del pathname ha associato dei permessi (in realtà li ha il relativo inode) che sono input all'algoritmo access control
  - di tali permessi l'algoritmo usa solo una tripletta (vedi slide successiva).

- **tutte le operazioni richiedono premesso search** (x) su tutte le directory nominate in tutti i pathname passati come parametro
  - compresa la directory corrente per pathname relativi
  - …**e in più richiedono permessi specifici**...

algoritmo per access control: selezione della tripletta

1. **UID della risorsa = EUID del processo**
   allora si usa solo (!) la tripletta **“user”**
   altrimenti...

2. **GID della risorsa = EGID o in supplumentary groups del processo**
   allora si usa solo (!) la tripletta **“group”**
   altrimenti...

3. si usa solo (!) la tripletta **“others”**

filesystem unix: permessi specifici richiesti per processi **non** privilegiati

- permessi specifici richiesti da operazioni su files
  - open create: w sulla directory (permessi del nuovo file stabiliti da parametro e umask)
  - open read: r sul file
  - open write append truncate: w sul file
  - execute: x sul file
  - link: w sulla directory destinazione
  - unlink: w sulla directory
- permessi specifici richiesti da operazioni su files e directory
  - chmod: uid del processo uguale a uid del inode
  - chown: uid del processo uguale a uid del inode
  - stat: -
- permessi specifici richiesti da operazioni su directory
  - mkdir: w sulla directory contenitore
  - rmdir: w sulla directory contenitore
  - readdir: r sulla directory da leggere

filesystem unix: permessi specifici richiesti processi privilegiati

nessuno

set-user-id bit (suid)

- molti comandi hanno bisogno dei diritti di “root” per funzionare correttamente ma gli utenti senza specifici diritti devono poterli comunque eseguire
  - alcuni esempi: ping, sudo, passwd, e molti altri!
- tali comandi hanno il bit “set user id” settato

```
pizzonia@pisolo$ cd /bin
pizzonia@pisolo$ ls -l ping
-rwsr-xr-x  1 root root 30764 Dec 22  2003 ping
```

- **un eseguibile con tale bit settato viene eseguito con EUID pari al proprietario del file indipendentemente da chi ne ha richiesto l’esecuzione**

set-group-id bit (sgid)

- il bit set-group-id si comporta in maniera analoga per l'EGID

```
-rwxr-sr-x  1 root tty      /usr/bin/wall
-rwxr-sr-x  1 root crontab  /usr/bin/crontab
```

- **usando set-uid e set-gid non necessariamente si “diventa root”, può bastare un utente o un gruppo che è proprietario di certi file necessari all'applicazione in questione**
- **questo aumenta la sicurezza del sistema**

minimalità dei diritti

- questo sistema di diritti ha una limitata espressività
- il principio della minimalità dei diritti è difficile da raggiungere con l'access control standard

- si usano altri strumenti come SELinux e AppAromor

unix - autenticazione e login

fasi del login

il login prevede 4 fasi

1. richiesta all'utente delle credenziali
   - es. username e password
2. verifica nel db degli utenti della correttezza delle credenziali
3. cambio di utenza
   - setreuid() + setregid()
4. execve della shell
   - e/o dell'ambiente grafico

db utente: approccio “standard”

- utenti e altri attributi in /etc/passwd
  - world readable
- gruppi in /etc/group
  - contiene anche la lista degli utenti appartenenti a ciascun gruppo
  - world readable
- passwords e altro in /etc/shadow e /etc/gshadow
  - leggibili solo a root

contenuto di /etc/passwd

- Login name
- Optional encrypted password
  - insicuro!!!!
  - vedi /etc/shadow
- Numerical user ID (root ha UID=0)
- Numerical group ID
- User name or comment field
- User home directory (es. /home/pizzonia)
- User command interpreter (es. /bin/bash)

contenuto di /etc/group per configurare supplementary groups

- group_name
- password (per il group managment)
  - the (encrypted) group password. If this field is empty, no password is needed.
- the numerical group ID
- user_list
  - all the group member's user names, separated by commas.

contenuto di /etc/shadow

- Login name (foreign key /etc/passwd)
- Encrypted password
  - idsalt$hashedpassword
  - id: codice per l'algoritmo
- Days since Jan 1, 1970 that password was last changed
- Days before password may be changed
- Days after which password must be changed
- Days before password is to expire that user is warned
- Days after password expires that account is disabled
- Days since Jan 1, 1970 that account is disabled
- A reserved field

comandi

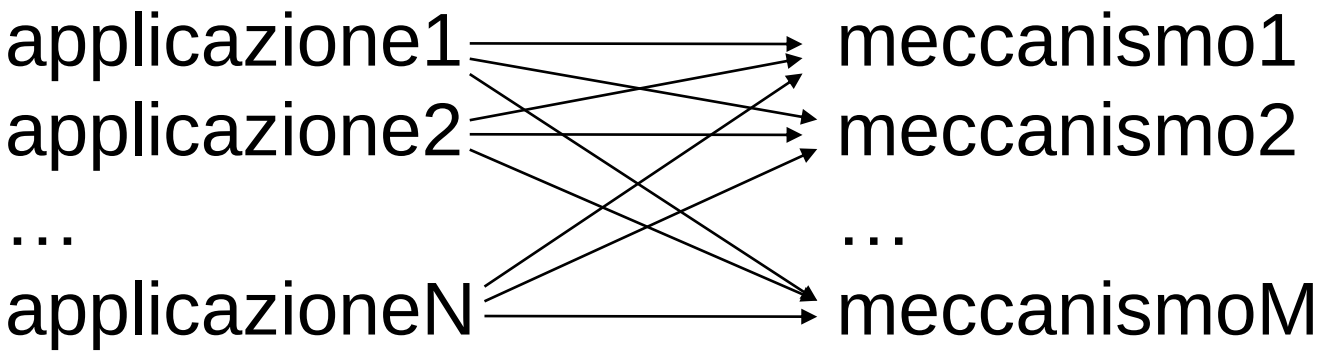
- l'editing a mano dei file passwd groups ecc. è possibile
  - …ma sconsigliato
  - bisogna rispettare rigidamente il formato dei file
    - gli amministratori smaliziati lo fanno
- comandi previsti per la gestione degli utenti e dei gruppi
  - useradd o adduser
  - userdel o deluser
  - usermod
  - groupadd o addgroup
  - groupdel o delgroup
  - groupmod
  - passwd
  - gpasswd

flessibilità nell'autenticazione

- molte sono le politiche e i meccanismi possibili
  - es. autenticazione locale (/etc/passwd, /etc/shadow)
  - es. autenticazione centralizzata (radius, active directory, ldap, nis)
  - es. diverse politiche: qualcosa che ho/so/sono/dove-sono
- molti sono i programmi che richiedono funzioni di autenticazione
  - es. programmi di sistema: atd, chfn, chsh, cron, cupsys, cvs, kcheckpass, kdm, kdm-np, kscreensaver, libcupsys2, login, passwd, ppp, samba, ssh, su, sudo, telnetd, xdm, xscreensaver
  - ma anche applicativi: mysql, apache, ecc.

flessibilità nell'autenticazione

- cosa succede se vogliamo cambiare la politica di autenticazione?
  - tutti i programmi che autenticano devono essere ricompilati con adeguato supporto
  - improponibile
- per un piena flessibilità sono necessarie NxM implementazioni



soluzione architetturale: PAM

- fattorizzazione della funzionalità dalle applicazioni
  - libreria condivisa
- configurabilità della libreria
- solo M implementazioni necessarie

- Pluggable Authentication Modules (PAM)
  - diffuso nel mondo unix
  - windows non ha una soluzione altrettanto flessibile

PAM: offre quattro “servizi”

- **auth**
  - autentica l'utente (normalmente tramite password ma non necessariamente!)
  - può assegnare delle credenziali al processo
    - non assegna UID e GID, usato per esempio per i tiket kerberos
- **account.** garantisce l'accesso per tutto ciò che non riguarda l'autenticazione. Esempio: accessi basati sul'ora, risorse di sistema, utente locale o remoto, ecc.
- **session.** logging o altre attività in apertura o chiusura di sessione.
- **password.** gestisce l'aggiornamento delle password.

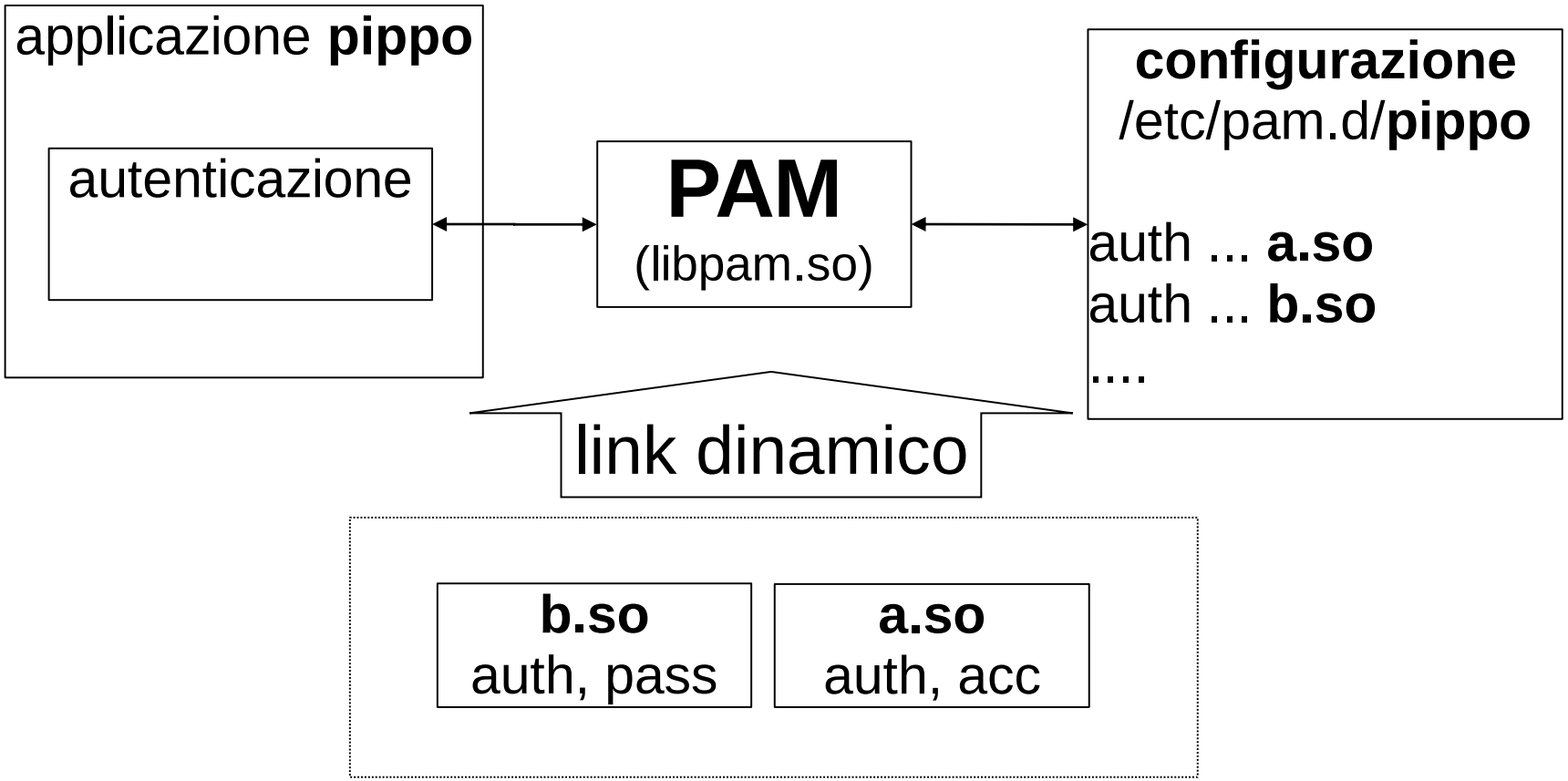
fasi del login e PAM

il login prevede 4 fasi

1. richiesta all'utente delle credenziali
   - es. username e password
2. verifica nel db degli utenti della correttezza delle credenziali
3. cambio di utenza
   - setreuid() + setregid()
4. execve della shell
   - e/o dell'ambiente grafico

PAM auth

PAM: architettura



moduli PAM (librerie dinamiche)

PAM: vantaggi per l'amministratore

- non c'è necessità di toccare il codice
- configurazione indipendente per ciascuna applicazione
- ampia scelta di meccanismi di autenticazione
  - uso di server esterni (es. radius, LDAP, active directory, ecc)
  - uso di dispositivi hardware (biometrici, token ecc.)
- possibilità di comporre vari metodi
  - multi-factor authentication

PAM: esempio di configurazione

```
/etc/pam.d/$ cat login
# PAM configuration for login
auth        requisite   pam_securetty.so
auth        required    pam_nologin.so
auth        required    pam_env.so
auth        required    pam_unix.so nullok
#auth       required        pam_permit.so
account     required    pam_unix.so
session     required    pam_unix.so
session     optional    pam_lastlog.so
password    required    pam_unix.so nullok
  obscure min=4 max=8
```

PAM: successo e/o fallimento

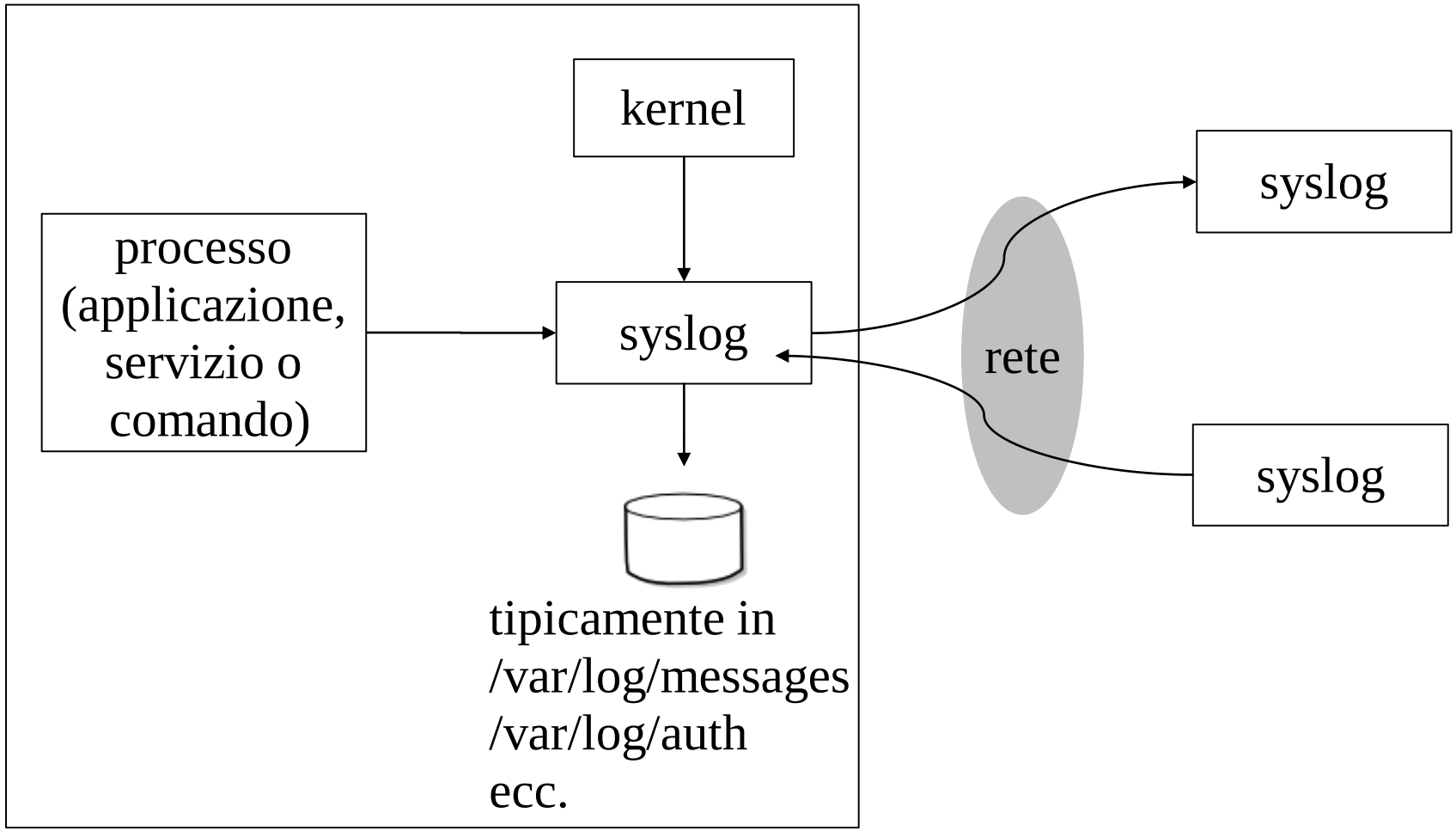
- ciascun modulo ritorna “successo” o “fallimento”
- **required, requisite**. se il modulo ritorna fallimento il login non ha successo.
  - se un “requisite” fallisce si termina la procedure immediatamente
- **sufficient**. il successo di un tale modulo è sufficiente per l'autenticazione (anche se c'è stato un fallimento precedente)
- **optional.** il valore di ritorno è ignorato.

- recenti versioni di PAM supportano un linguaggio più espressivo

unix - logging

syslog

- syslog è il servizio di logging normalmente attivo su ogni macchina Unix



priorità

severità o livello

- debug
- info: messaggio informativo
- notice: situazione normale ma significativa
- warning, *warn*
- err, *error*
- crit
- alert: una azione deve essere presa immediatamente
- emerg, *panic*

categorie di messaggi
(facilities o selettori)

- auth, authpriv, cron, daemon, ftp, kern, lpr, mail, mark, news, security (same as auth), syslog, user, uucp, local0...local7
- largamente obsoleti

configurazione di syslog

- /etc/(r)syslog.conf
- /etc/(r)syslog.d/….
- ciascuna linea segue il formato
  <selettore> <azione>
- esempio: log di tutti i messaggi della categoria “auth” e importanza maggiore uguale a “notice” dentro /var/log/auth

```
auth.notice  /var/log/auth

auth.!err     #variante: importanza minore di err
auth.=warn     #variante: importanza uguale a warn
*.err  #variante: qualsiasi categoria purché >err
```

configurazione di syslog: esempi

```
cron.*                          -/var/log/cron.log
daemon.*                        /var/log/daemon.log
kern.*                          /var/log/kern.log
auth,authpriv.*                     /var/log/auth.log
*.*;auth,authpriv.!=none                /var/log/syslog


*.=info;*.=notice;*.=warn;\
  auth,authpriv.none;\
  cron,daemon.none;\
  mail,news.none
  /var/log/messages
```

remote logging

- (r)syslogd può comportarsi da server
- (r)syslogd può comportarsi da client
  - è un particolare tipo di azione, es.:
    *.warn @193.204.161.48
    auth.crit @syslogserver.dia.uniroma3.it
- vantaggi
  - fare il log di molte macchine su una sola può snellire moltissimo i successivi auditing
  - un hacker per modificare i log su un syslog server deve penetrare anche in tale macchina
  - il syslog server può essere ben protetto
- chiunque può aggiungere righe al log!!
  - protocollo senza autenticazione, basato su UDP