

Securitatea sistemelor de calcul

Controlul accesului

5 octombrie 2012

Politică și mecanism

O *politică de securitate* e o afirmație despre acțiunile permise/nepermise.

Un *mecanism de securitate* e o metodă, unealtă sau procedură pentru asigurarea (enforcement) unei politici de securitate.

Bishop, Computer Security: Art and Science

⇒ se impune verificarea corectitudinii mecanismului
mecanismul poate fi:

- sigur (dacă nu permite stări nepermise de politică);
- precis (permite exact ce specifică politica),
- larg (broad) (permite mai mult decât politica)

Controlul accesului

Mecanism de a permite sau interzice accesul unei entități la o resursă.

“principal” /subiect → solicitare → guard → obiect

Controlul accesului constă în doi pași:

autentificare: Cine a făcut cererea de acces ?

autorizare Are subiectul s drept de acces la obiectul o ?

Controlul accesului

Distingem:

- o mulțime de subiecți S
- o mulțime de obiecte O
- o mulțime de moduri de acces A .

Cel mai elementar: $A = \{observe, alter\}$. Prea simplu, de obicei...

Rafinând, modelul Bell-LaPadula propune:

$$A = \{execute, read, append, write\}.$$

Cum se exprimă în funcție de permisiunile elementare ? Când sunt utile distincțiile între aceste moduri ?

Access Control Matrix

Cea mai simpla/generala organizare: matrice de control al accesului

Doua dimensiuni: subiecti și obiecte.

– în fiecare intrare $S \times O$: mulțime de drepturi/permisiuni

– un subiect poate fi și obiect (e.g. un proces):

dreptul de a citi/scrie (comunica) / executa un alt proces

	file1	/etc/passwd	/bin/rlogin
Alice	r, w	r	x
Bob	r	-	r, x

Drepturi și atenuarea lor

copy right (grant right): dreptul de a acorda permisiuni altora

own right (admin): dreptul de a-si acorda permisiuni sie insusi

Principiul atenuării privilegiilor:

Un subiect nu poate acorda altora drepturi pe care nu le posedă

Q: În UNIX, proprietarul unui fișier poate să dea altuia (group/others) drepturi de citire asupra fișierului, chiar dacă el însuși nu le posedă.

Se violează principiul de mai sus ?

Semantica identificatorilor UID în UNIX

Un *proces* are (în versiuni noi) *trei* identificatori denotând utilizatorul:

- *real* user ID: proprietarul procesului
- *effective* user ID: determină permisiunile de acces
- *saved* user ID: folosit pentru a reveni la un UID anterior

Normal: ruid = euid = utilizatorul care lansează procesul

Excepție: euid = proprietarul *fișierul executabil încărcat*, când acesta are bitul **s** (setuid) setat \Rightarrow rularea cu alte privilegii (ex. superioare)

(similar pentru identificatorii de grup)

Q1: De ce sunt necesare funcții pentru a manipula uid la execuție?

Semantica identificatorilor UID în UNIX

Un *proces* are (în versiuni noi) *trei* identificatori denotând utilizatorul:

- *real* user ID: proprietarul procesului
- *effective* user ID: determină permisiunile de acces
- *saved* user ID: folosit pentru a reveni la un UID anterior

Normal: ruid = euid = utilizatorul care lansează procesul

Excepție: euid = proprietarul *fișierul executabil încărcat*, când acesta are bitul **s** (setuid) setat \Rightarrow rularea cu alte privilegii (ex. superioare)
(similar pentru identificatorii de grup)

Q1: De ce sunt necesare funcții pentru a manipula uid la execuție?

Q2: De ce nu cade salvarea vechiului UID în sarcina programatorului?

Apelurile setuid / seteuid

setuid(val)

- dacă euid e 0 (root), setează ruid=euid=val (și saved uid)
⇒ identificatorul / privilegiile sunt setate *irevocabil*
- altfel (euid \neq 0) poate seta doar euid = val dacă val == ruid
ruid și saved uid rămân neschimbate

Q3: care sunt limitările dacă există doar acest apel?

seteuid(val)

- permis doar dacă euid == 0
- sau dacă val e una din cele 3 valori (euid/ruid/saved)
setează *doar* euid, nu modifică ruid și saved uid.
⇒ modificările sunt *reversibile* prin alt apel seteuid

Rezultate fundamentale

E simplu să proiectăm un sistem corect de control al accesului ?

Rezultate fundamentale

E simplu să proiectăm un sistem corect de control al accesului ?

Def: Un sistem e *sigur* în raport cu un anumit drept (al unui subiect asupra unui obiect), dacă nu există o secvență de tranziții (operații) prin care dreptul să poată fi adăugat, presupunând că nu există inițial

Thm: Siguranța unui sistem (arbitrar) într-o stare, relativ la un anumit drept de acces e *nedecidabilă*.

Dem: o mașină Turing se poate reduce la (codifica în) acest sistem
– în esență, datorită posibilității de a crea obiecte noi

Subclase mai simple de sisteme sunt însă decidabile

– dacă se permit doar operații simple

– dacă nu au condiții multiple și sunt monotone (se permite crearea dar nu și distrugerea de subiecți/obiecte)

Tipuri de control al accesului

Discretionary Access Control (DAC)

permite *utilizatorilor individuali* (tipic: owner) să seteze mecanismul prin care se permite/interzice accesul

Mandatory Access Control (MAC)

mecanismul de acces e determinat de sistem, nu poate fi schimbat de utilizatori

tipic: bazat pe un set de reguli (rule-based access control)

Q: care sunt avantajele si dezavantajele celor două categorii ?

Role-Based Access Control (RBAC)

politică determinată de sistem, în funcție de *rolul* activ al unui *subiect ierarhie* de roluri (unele pot fi incluse în altele)

poate modela diverse cerințe, de ex. *separation of duty*

Mecanisme: Access Control Lists (ACL)

Reprezentare a matricii de acces în care *fiecare obiect* are asociată o listă de subiecți și permisiunile lor

– caz simplu: permisiunile sub UNIX

Exemplu cu set mai bogat de permisiuni: Andrew File System

read

list (pentru director: conținutul)

insert (fișier nou în director)

delete (fișier din director)

write

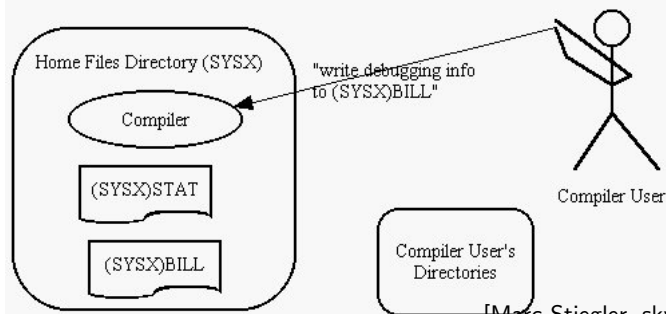
lock (poate folosi apeluri *lock* în director)

administer

O problema clasica: Confused Deputy

Norman Hardy, The Confused Deputy(or why capabilities might have been invented), ACM SIGOPS Operating Systems Review, 22(4), 1988

Never Separate An Object From Its Authority



The Confused Deputy

Who is to blame?

- The code to deposit the debugging output in the file named by the user?
- Must the compiler check to see if the output file name is in another directory?
- Should the compiler check for directory name SYSX?
- Should the compiler check for the name (SYSX)BILL?

Mecanisme: Capabilities

Înțeles oarecum diferit: *în SO*, desemnează un *identificator* care denotă un obiect *și* drepturile asociate cu acesta:

ex. descriptor de fișier (la deschidere se stabilește și modul de acces)

În *securitate*, o capabilitate e o *listă de drepturi* ale unui subiect (corespunde cu un rând din matricea de control al accesului)

Exemplu: POSIX/Linux Capabilities `capability.h`

Multilevel Security (MLS)

Inspirată din domeniul militar. Definește *nivele de securitate*
ex. public \leq restricționat \leq confidențial \leq strict secret
evtl: compartimentare după domeniul de interes (*need-to-know basis*)
Mulțimea atributelor de securitate e ordonată într-o *latice*

Modelul Bell-La Padula: urmărește *confidențialitatea*. 2 reguli:
no read up: un subiect nu poate citi peste nivelul său de securitate
no write down: nu poate scrie (dezvălui) ceva sub nivelul propriu

Modelul Biba: urmărește *integritatea*. Reguli duale:
interzisă scrierea deasupra nivelului propriu și citirea (folosirea) datelor aflate sub acest nivel: ambele pot corupe integritatea

Pentru a atinge ambele, în practică un subiect își poate să-și coboare voluntar nivelul de privilegiu

Integritate: de la practica la teorie

Principii de securitate în dezvoltarea de soft comercial (Lipner):

- *utilizatorii* nu își vor scrie propriile programe, ci vor folosi soft de producție existent
- programatorii vor dezvolta și testa pe alt sistem decât cel de producție
- instalarea unui program va presupune un proces special
- acest proces trebuie supus la *control* și *audit*
- managerii/auditorii vor avea acces și la starea de sistem și la log-uri

Principii:

- separation of duty
- separation of function
- audit/accountability