CSE 380 Computer Operating Systems

Instructor: Insup Lee and Dianna Xu

University of Pennsylvania Fall 2003 Lecture Note: Protection Mechanisms

Policy vs. Mechanism

Access control policy is a *specification*

- Given in terms of a model of the system
- Subjects: do things (i.e. a process writes to files)
- Objects: are passive (i.e. the file itself)
- Actions: what the subjects do (i.e. read a string from a file)
- Rights: describe authority (i.e. read or write permission)

□ Mechanisms are used to *implement* a policy

- Example: access control bits in Unix file system & OS checks
- Mechanism should be general; ideally should not constrain the possible policies.
- Complete mediation: every access must be checked



Example Reference Monitors

- Operating Systems
 - File system
- Memory (virtual memory, separate address spaces)
- Firewalls
 - Regulate network access
- Java Virtual Machine
 - Regulates Java programs' resource usage

A[s][o]	Obj ₁	Obj ₂	 Obj _N
Subj ₁	{r,w,x}	{r,w}	 0
Subj ₂	{w,x}	0	
			 a
Subj _M	{X}	{r,w,x}	 {r,w,x



Model for resource Protection

- A Protection System is composed of
 - A Protection System is composed of set of subjects: processes executing in a specific protection domain
 set of objects: all the passive elements of the system plus all the subjects
 - set of rules specifying the protection policy
- Protection Domain: Set of rights a process has at any given time
 Protection state is checked for each access of an object, X, by a subject, S
- Protection state can be conceptualized as an *access matrix*.
- A[S,X] is a set that describes the access rights held by subject S to object X.



Rights and Actions

- Besides read, write, execute actions there are many others:
- Ownership
- Creation
 - New subjects (i.e. in Unix add a user)
 - New objects (i.e. create a new file)
 - New rights: Grant right r to subject s with respect to object o (sometimes called delegation)
- Deletion of
 - Subjects Objects
 - Rights (sometimes called revocation)

Protecting the Reference Monitor

It must not be possible to circumvent the reference monitor by corrupting it

Mechanisms

- Type checking
- Software fault isolation: rewrite memory access instructions to perform bounds checking
- User/Kernel modes
- Segmentation of memory (OS resources aren't part of virtual memory system)

Storing the Access Control Matrix

□ Subjects >> # users

- A row can correspond to a protection domain
- · Each subject runs within a protection domain
- Example: User-ID and Group-ID in Unix determine domain
 Objects >> # files
- Potentially could have permissions on any resource

□ The matrix is typically sparse

Store only non-empty entries

Α	ccess Control Lists					
A[s][o]	Obj ₁	Obj ₂		Obj _N		
Subj ₁	{r,w,x}	{r,w}		0		
Subj ₂	{w,x}	0		{r}		
Subj _M	{x}	{r,w,x}		{r,w,x}		

For each object, store a list of (Subject , Rights) pairs.

Access Control Lists

- Besolving queries is linear in length of the list
- Revocation w.r.t. a single object is easy
- $\hfill\square$ "Who can access this object?" is easy

Useful for auditing

- Lists could be long
 - Factor into groups (lists of subjects)Give permissions based on group
 - Give permissions based on group
 - Introduces consistency question w.r.t. groups
- Authentication critical

11

• When does it take place? Every access would be expensive.

12

A[s][0]	Obj ₁	Obj ₂	 Obj _N
Subj ₁	{r,w,x}	{r,w}	 0
Subj ₂	{w,x}	0	 {r}
Subj _M	{x}	{r,w,x}	 {r,w,x}



Storing Capabilities Securely

- □ Special hardware: tagged words in memory
- Can't copy/modify tagged words
- □ Store the capabilities in protected address space
- $\hfill\square$ Could use static scoping mechanism of safe
 - programming languages.
 - Java's "private" fields
- Could use cryptographic techniques
 - OS kernel could sign (Object, Rights) pairs using a private key
 - Any process can verify the capability

Unix Security

- Each user has a unique 16-bit UID
 UID of root/superuser is 0
- □ Each user can belong to a group, each group has a unique 16-bit GID
- Protection domain of a process is determined by the (UID,GID) of the user that owns the process
- Every file has

15

- UID and GID of the owner
- Protection bits that can be set/changed by the owner
- Devices handled as files (e.g. /dev/tty, /dev/lp)
- □ 9 bits specifying allowed read(r)/write(w)/execute(x) access for the owner, group, and everyone else
 - E.g. rw-r---- means owner can read/write and group can read

SETUID

- □ How to give temporary access to privileged resources?
- □ E.g. /dev/lp is owned by printer daemon (or by root), other processes need to write to it to send jobs to printer, but you do not want to set permission to rwxrwxrwx
- □ Solution: Each file/device has a SETUID bit
- □ When an executable program P with SETUID bit set to 1 is executed by a process Q, the protection domain of Q is changed to (UID,GID) of P (i.e. the owner of P)
 - If P's SETUID bit is 0, then protection domain of Q does not change

Sample Scenario

- /dev/lp is owned by root with protection rw-----This is used to access the printer
- □ /bin/lp is owned by root with --x--x--x with SETUID=1
- User A issues a print command
- □ Shell (running with A's UID and GID) interprets the command and forks off a child process, say, P
- Process P has the same UID/GID as user A
- □ Child process P executes exec("/bin/lp",...)
- □ Now P's domain changes to root's UID
- Consequently, /dev/lp can be accessed to print
- U When /bin/lp terminates so does P

17

□ Parent shell never got the access to /dev/lp