

CISC 7310X

C15a The Protection Problem and Access Matrix

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook via the publisher of the textbook

Outline

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Example implementations in UNIX and Windows

Security and Protection

- Security systems
 - authenticate system users to protect the integrity of program code, data, and the physical resources of the computer system.
 - prevent unauthorized access, malicious destruction or alteration of data, and accidental introduction of inconsistency.
- Protection mechanisms
 - controls the access of programs, processes, or users to the resources defined by a computer system.

Protection Model: Process & Objects

- A computer consists of a collection of objects, hardware or software
 - Each object has a unique name and can be accessed through a well-defined set of operations
 - Processes carry out the operations
 - Hardware objects (such as devices)
 - Software objects (such as files, programs, semaphores)
- Process should only have currently required types of access to currently required objects to complete its task
 - the least-privilege principle
 - the need-to-know principle

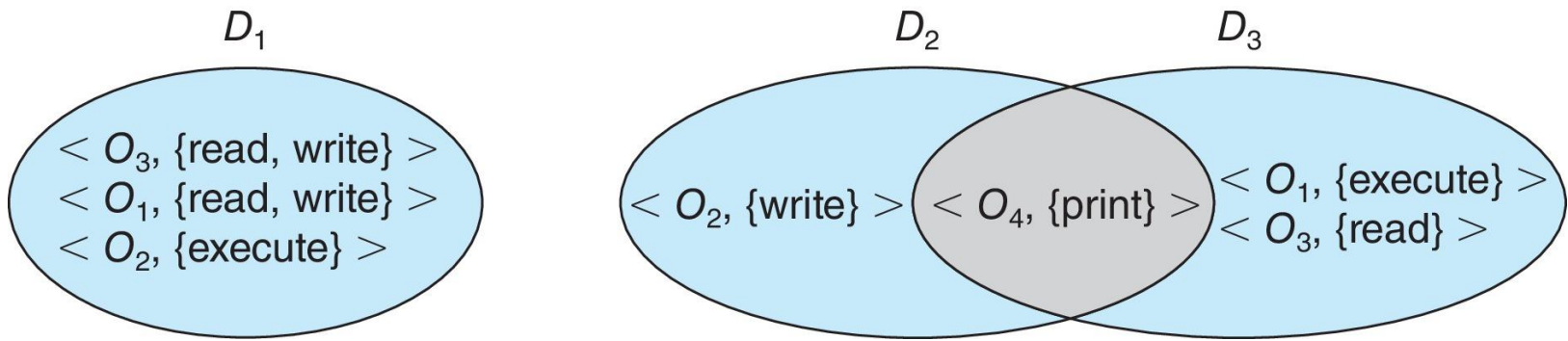
Domain of Protection

- A process may operate within a protection domain that specifies the resources that the process may access.
- Example
 - Ability to execute an operation on an object is an access right
 - Example operation: read, write, execute, list
 - A domain can be defined as a collection of access rights
 - each of which is an ordered pair <object-name, rights-set>.
 - A process is associated with a domain
 - Associations can be **static** or **dynamic**
 - If dynamic, processes can **switch domains**

Examples of Protection Domain

- Domain = set of access-rights
- Access-right = $\langle \text{object-name}, \text{rights-set} \rangle$
- Rights-set is a subset of all valid operations that can be performed on the object
- Example
 - 3 protection domains (D_1 , D_2 , and D_3) below

Example: 3 Protection Domains



Representing Protection

Domain: Access Matrix

- View protection as a matrix, called Access Matrix or Access Control Matrix
- Rows represent domains
- Columns represent objects
- $\text{Access}(i, j)$ is the set of operations that a process executing in Domain_i can invoke on Object_j

Access Matrix: Example

- A process associated with D_i will have the specified rights for the specified objects

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- Associate a process with a domain (a process is in/is executing in/enters the domain)
- If a process in Domain D_i tries to do "op" on object O_j , then "op" must be in the access matrix
- User who creates object can define access column for that object

Access Matrix: Policy & Mechanism

- Access matrix design separates mechanism from policy
- Mechanism
 - Operating system provides access-matrix + rules
 - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
- Policy
 - User dictates policy
 - Who can access what object and in what mode

Access Matrix: Dynamic Protection

- Can be expanded to dynamic protection (domain switch)
 - Operations to add, delete access rights
 - Special access rights:
 - *transfer* - switch from domain D_i to D_j
 - *owner* of O_i
 - *copy op* from O_i to O_j (denoted by "*")
 - *control* - D_i can modify D_j access rights
 - *Copy* and *Owner* applicable to an object
 - *Control* applicable to domain object

Access Matrix Domain Switch: Example

- A process in D_2 can enter/switch to D_3 or D_4

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Copy Right

- The ability to copy an access right from one domain (or row) of the access matrix to another
- denoted by an asterisk (*) appended to the access right.
- The copy right allows the access right to be copied only within the column (that is, for the object) for which the right is defined.

Access Matrix with Copy Rights: Example

- A process executing in domain D_2 can copy the read operation into any entry associated with file F_2 .

object \ domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object \ domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Owner Right

- If $\text{access}(i,j)$ includes the owner right, then a process executing in domain D_i can add and remove any right in any entry in column j .

Access Matrix with Owner Rights: Example

- the access matrix of (a) can be modified to the access matrix (b).

object \ domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object \ domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Control Right

- The copy and owner rights allow a process to change the entries in a column.
- The control right is applicable only to domain objects, i.e., to change the entries in a row (or a domain)
- Example
 - See below
 - A process is executing in domain D_2 can change D_4

object \ domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	read write		read write		switch			

object \ domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- Generally, a sparse matrix
 - How much memory do we need to naively/directly implement an access matrix?
- Examples
 - Global table
 - Access list (access-control list)
 - Capability list
 - Lock key

Global Table

- Store ordered triples $\langle \text{domain}, \text{object}, \text{rights-set} \rangle$ in table
- A requested operation M on object O_j within domain D_i \rightarrow search table for $\langle D_i, O_j, R_k \rangle$
 - with $M \in R_k$
- But table could be large \rightarrow won't fit in main memory
 - How big?
- Difficult to group objects (consider an object that all domains can read)

Global Table: Example

- Given 3 domains and 3 files:
 - r,w,x,o = read, write, execute, own

	File1	File2	File3
D1	rx	r	rwo
D2	rwxo	r	
D3	rx	rwo	w

- Q: how to store this as a global table?

Global Table: Example

- Given 3 domains and 3 files:
 - r,w,x,o = read, write, execute, own

	File1	File2	File3
D1	rx	r	rwo
D2	rwxo	r	
D3	rx	rwo	w

- Global table (3 columns, 6 rows)
 - $\langle D1, \text{File1}, rx \rangle, \langle D1, \text{File2}, r \rangle, \langle D1, \text{File3}, rwo \rangle, \langle D2, \text{File1}, rwxo \rangle, \langle D2, \text{File2}, r \rangle, \langle D3, \text{File1}, rx \rangle, \langle D3, \text{File2}, rwo \rangle, \langle D3, \text{File3}, w \rangle$

Access Lists (Access-Control Lists) for Objects

- Each column implemented as an access list for one object
- Resulting per-object list consists of ordered pairs $\langle \text{domain}, \text{rights-set} \rangle$ defining all domains with non-empty set of access rights for the object
- Easily extended to contain default set \rightarrow If $M \in \text{default set}$, also allow access

Access Lists: Example

- Each column implemented as an access list for one object

	File1	File2	File3
D1	rx	r	rwo
D2	rwxo	r	
D3	rx	rwo	w

- File1: {<D1, rx>, <D2, rwxo>, <D3, rx>}
- File2: {<D1, r>, <D2, r>, <D3, rwo>}
- File3: {<D1, rwo>, <D3, w>}

Access List and Capability List

- Access list for objects
 - Each column = Access list for one object
Defines who can perform what operation
 - Domain 1 = Read, Write
 - Domain 2 = Read
 - Domain 3 = Read
- Capability list for domains
 - Each Row = Capability List for each domain, what operations allowed on what objects
 - Object F1 - Read
 - Object F4 - Read, Write, Execute
 - Object F5 - Read, Write, Delete, Copy

Capability List for Domains

- Instead of object-based, list is domain based
- **Capability list** for domain is list of objects together with operations allows on them
- Object represented by its name or address, called a **capability**
- Execute operation M on object O_j , process requests operation and specifies capability as parameter
 - Possession of capability means access is allowed
- Capability list associated with domain but never directly accessible by domain
 - Rather, protected object, maintained by OS and accessed indirectly
 - Like a "secure pointer"
 - Idea can be extended up to applications

Capability Lists: Example

- Capability list for domain is list of objects together with operations allows on them

	File1	File2	File3
D1	rx	r	rwo
D2	rwxo	r	
D3	rx	rwo	w

- D1: { <file1, rx>, <file2, r>, <file3, rwo> }
- D2: { <file1, rwxo>, <file2, r> }
- D3: { <file1, rx>, <file2, rwo>, <file3, w> }

Lock Key

- Compromise between access lists and capability lists
- Each object has list of unique bit patterns, called **locks**
- Each domain as list of unique bit patterns called **keys**
- Process in a domain can only access object if domain has key that matches one of the locks

Comparison of Implementations

- Many trade-offs to consider
 - Global table is simple, but can be large
 - Access lists correspond to needs of users
 - Determining set of access rights for domain non-localized so difficult
 - Every access to an object must be checked
 - Many objects and access rights -> slow
 - Capability lists useful for localizing information for a given process
 - But revocation capabilities can be inefficient
 - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Implementation

- Most systems use combination of access lists and capabilities
 - First access to an object -> access list searched
 - If allowed, capability created and attached to process
 - Additional accesses need not be checked
 - After last access, capability destroyed
 - Consider file system with ACLs per file

Revocation of Access Rights

- Various options to remove the access right of a domain to an object
 - Immediate vs. delayed
 - Selective vs. general
 - Partial vs. total
 - Temporary vs. permanent

Revocation of Access Rights in Access List

- **Access List** - Delete access rights from access list
 - Simple - search access list and remove entry
 - Immediate, general or selective, total or partial, permanent or temporary

Revocation of Access Rights in Capability List

- **Capability List** - Scheme required to locate capability in the system before capability can be revoked
 - **Reacquisition** - periodic delete, with require and denial if revoked
 - **Back-pointers** - set of pointers from each object to all capabilities of that object (Multics)
 - **Indirection** - capability points to global table entry which points to object - delete entry from global table, not selective (CAL)
 - **Keys** - unique bits associated with capability, generated when capability created
 - Master key associated with object, key matches master key for access
 - Revocation - create new master key
 - Policy decision of who can create and modify keys - object owner or others?

Implementing Protection

Domain: UNIX

- Domain = user-id (or group-id, or "public")
- Domain switch accomplished via file system
 - Each file has associated with it a domain bit (setuid bit)
 - When file is executed and setuid = on, then user-id is set to owner of the file being executed
 - When execution completes user-id is reset
- Domain switch accomplished via passwords
 - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
 - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

A Sample UNIX Directory Listing

```

-rw-rw-r-- 1 pbg staff 31200 Sep 3 08:30 intro.ps
drwx----- 5 pbg staff 512 Jul 8 09:33 private/
drwxrwxr-x 2 pbg staff 512 Jul 8 09:35 doc/
drwxrwx--- 2 pbg student 512 Aug 3 14:13 student-proj/
-rw-r--r-- 1 pbg staff 9423 Feb 24 2003 program.c
-rwxr-xr-x 1 pbg staff 20471 Feb 24 2003 program
drwx--x--x 4 pbg faculty 512 Jul 31 10:31 lib/
drwx----- 3 pbg staff 1024 Aug 29 06:52 mail/
drwxrwxrwx 3 pbg staff 512 Jul 8 09:35 test/

```

Permissions

user & group

size

modify time

name

number of hardlinks

Access Lists in UNIX

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) owner access	7	⇒	1 1 1 RWX
b) group access	6	⇒	1 1 0 RWX
c) public access	1	⇒	0 0 1 RWX

Access Groups in UNIX

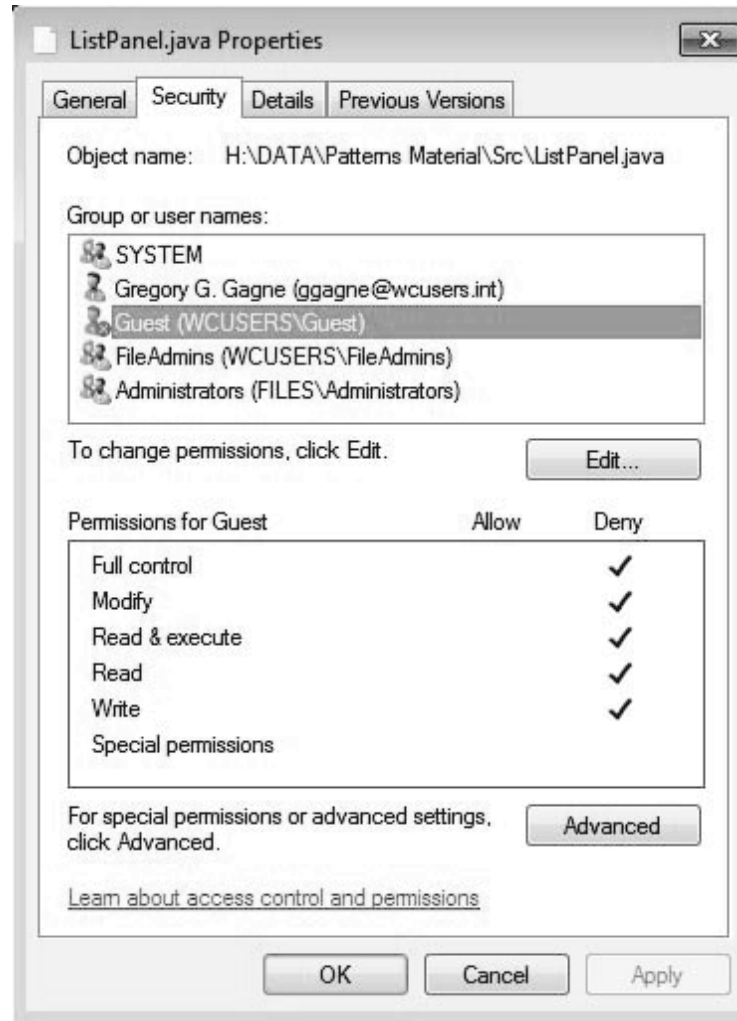
- Ask administrator/manager to create a group (unique name), say *G*, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

```
owner  group  public
  \      |      /
   chmod 761 game
```

Attach a group to a file

```
chgrp      G      game
```

Windows Access-Control List Management



Questions?

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Example implementations in UNIX and Windows