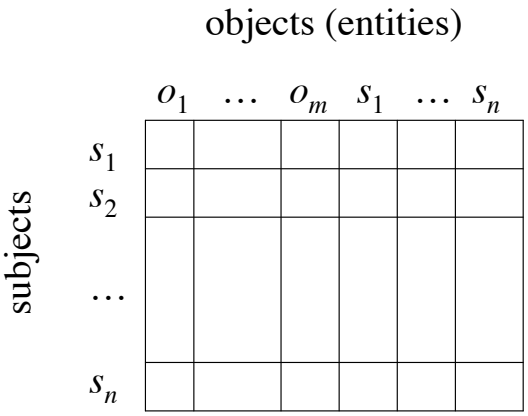
ECS 289M Lecture 2

April 3, 2006

Overview

- Protection state of system
  - Describes current settings, values of system relevant to protection
- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

Description



- Subjects $S = \{ s_1,\ldots,s_n \}$
- Objects $O = \{ o_1,\ldots,o_m \}$
- Rights $R = \{ r_1,\ldots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

Example 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Example 2

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights +, –, *call*

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

Boolean Expression Evaluation

- ACM controls access to database fields
  - Subjects have attributes
  - Verbs define type of access
  - Rules associated with objects, verb pair
- Subject attempts to access object
  - Rule for object, verb evaluated, grants or denies access

Example

- Subject annie
  - Attributes role (artist), groups (creative)
- Verb paint
  - Default 0 (deny unless explicitly granted)
- Object picture
  - Rule:

    paint: ‘artist’ in subject.role and
    ‘creative’ in subject.groups and
    time.hour ≥ 0 and time.hour < 5

ACM at 3AM and 10AM

At 3AM, time condition met; ACM is:

	… picture …		
… annie …			
		paint	

At 10AM, time condition not met; ACM is:

	… picture …		
… annie …			

History

Database:

name	position	age	salary
Alice	teacher	45	$40,000
Bob	aide	20	$20,000
Cathy	principal	37	$60,000
Dilbert	teacher	50	$50,000
Eve	teacher	33	$50,000

Queries:

1.sum(salary, “position = teacher”) = $140,000

2.sum(salary, “age > 40 & position = teacher”) should not be answered (deduce Eve’s salary)

ACM of Database Queries

O_i = { objects referenced in query i }; let $|O| = n$ (3, here)

$f(o_i)$ = { read } for $o_j \in O_i$, if $|\cup_{j = 1,\ldots,i} O_j| \neq 2$, n–1

$f(o_i) = \varnothing$ for $o_j \in O_i$, otherwise

1. O_1 = { Alice, Dilbert, Eve }, so $|O_1| = n$, and:

   A[asker, Alice] = f(Alice) = { read }

   A[asker, Dilbert] = f(Dilbert) = { read }

   A[asker, Eve] = f(Eve) = { read }

   and query can be answered

But Query 2

From last slide:

$f(o_i)$ = { read } for $o_j \in O_i$, if $|\cup_{j = 1,\ldots,i} O_j| \neq 2$, n–1

$f(o_i) = \varnothing$ for $o_j \in O_i$, otherwise

2. O_2 = { Alice, Dilbert } but | $O_1 \cup O_2$ | = n–1, so

A[asker, Alice] = f(Alice) = $\varnothing$

A[asker, Dilbert] = f(Dilbert) = $\varnothing$

and query cannot be answered

State Transitions

- Change the protection state of system
- |– represents transition
  - X_i |– $_\tau$ X_{i+1}: command τ moves system from state X_i to X_{i+1}
  - X_i |– * X_{i+1}: a sequence of commands moves system from state X_i to X_{i+1}
- Commands often called *transformation procedures*

Primitive Operations

- **create subject** *s*; **create object** *o*
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** *s*; **destroy object** *o*
  - Deletes row, column from ACM; deletes column from ACM
- **enter** *r* **into** *A*[*s*, *o*]
  - Adds *r* rights for subject *s* over object *o*
- **delete** *r* **from** *A*[*s*, *o*]
  - Removes *r* rights from subject *s* over object *o*

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** *s*
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** *o*
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Add Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: **enter** *r* **into** $a[s,o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Delete Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** r **from** $a[s,o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** s
- Postconditions:
  - $S' = S - \{ s \}$, $O' = O - \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** *o*
- Postconditions:
  - $S' = S$, $O' = O - \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
   create object f;
   enter own into A[p, f];
   enter r into A[p, f];
   enter w into A[p, f];
end
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

Conditional Commands

- Let *p* give *q r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
         enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

Biconditional Commands

- Let *p* give *q* *r* and *w* rights over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant•read•file•2(p, f, q)
    if own in A[p, f] and c in A[p, q]
    then
         enter r into A[q, f];
         enter w into A[q, f];
end
```


General Form of Commands

- Conditional part
  - At most one “if” allowed
  - “If” must be first thing in command
  - Only “and”s allowed in “if” statement
  - If condition(s) in “if” are false, body of command *not* executed
- Body
  - May contain commands and/or primitive operations
  - May *not* contain “if”s (embed them in called commands)

Example: Invalid Command

```
command create•file(p, q, r)
   create object o;
   if r in A[p, q] then
      enter r into A[p, o];
end
```


Example: Valid Command

```
command add•r•right(o, p, q, r)
   if r in A[p, q] then
      enter r into A[p, o];
end
command create•file(p, q, r)
   create object o;
  add•r•right(o, p, q, r);
end
```

Copy Right

- Allows possessor to give rights to another
- Often attached to a right, so only applies to that right
  - *r* is read right that cannot be copied
  - *rc* is read right that can be copied
- Is copy flag copied when giving *r* rights?
  - Depends on model, instantiation of model

Own Right

- Usually allows possessor to change entries in ACM column
  - So owner of object can add, delete rights for others
  - May depend on what system allows
    - Can't give rights to specific (set of) users
    - Can't pass copy flag to specific (set of) users

Attenuation of Privilege

- Principle says you can't give rights you do not possess
  - Restricts addition of rights within a system
  - Usually *ignored* for owner
    - Why? Owner gives herself rights, gives them to others, deletes her rights.

Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions