Access Control Matrix

- Protection state of system
  - Part of the system state (contents of memory locations, registers, etc.)
  - 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

1

Protection state

- State transitions change the system and therefore the protection state
- From all the states P of system, only those in Q considered safe (acceptable)
- **Security Policy** characterizes the states in Q
- **Security Mechanisms** prevent system from entering states in P-Q

2

Access Control Matrix

- Used to indicate **who** is allowed to do **what** to/with **whom** on the system.
- Who is **who** ?
  - **Subject** is what we call active entities (processes, users, other computers) that want to “do something”
- The **what** the subject does with the object can be just about anything, and it may be multi-part.
  - The set of rights in a cell specify the access of the subject (row) to the object (column)
  - Typical manipulations include
    - READ, MODIFY, CREATE, CHANGE, DELETE

3

Object

- **Object** is what we call the broad category of items which can be manipulated or accessed
- Usually objects are passive, for example:
  - File
  - Directory (or Folder)
  - Memory segment
- An entity might be considered a **subject** in some circumstances but in other ones be an **object** on which it is possible to perform operations such as
  - kill
  - suspend
  - resume

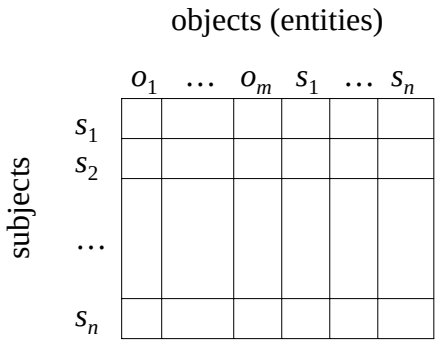
4

Operations can require multiple access rights

- Example: User *parisi* (subject) wishes to copy (manipulate) the file */etc/passwd* (object1) to his home *directory* (object2)
- On many systems it requires access checks as in the following:
  - *parisi* needs permission to read OBJECT1
  - *parisi* needs permission to create a new file (OBJECT3) in OBJECT2
  - *parisi* needs permission to append to (or write to) OBJECT3

5

Description of ACM



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

6

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*

7

Example 2

- Procedures (subjects) *inc_ctr*, *dec_ctr*, *manage*
- Variable (object) *counter*
- Rights *+*, *–*, *call*

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

8

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 |– * Y: a sequence of commands moves system from state X_i to Y
- Commands often called *transformation procedures*

9

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*

10

Create Subject

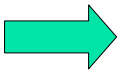
- Precondition: $s \notin S$
- Primitive command: **create subject** s
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$,
  - $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

11

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s

	p	q
x	r_1	r_1,r_2
y		r_1



	p	q
x	r_1	r_1,r_2
y		r_1
s		

12

Create Object

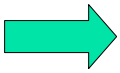
- Precondition: $o \notin O$
- Primitive command: **create object** o
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

13

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o

	p	q
x	r_1	r_1,r_2
y		r_1



	p	q	**o**
x	r_1	r_1,r_2	
y		r_1	

14

Add Right

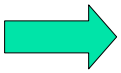
- Precondition: $s \in S$, $o \in O$
- Primitive command: enter r into $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x,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

15

Add Right

- Precondition: $p \in S$, $y \in O$
- Primitive command: **enter** r **into** $a[p, y]$

	p	q
x	r_1	r_1,r_2
y		r_1



	p	q
x	r_1	r_1,r_2
y	**r**	r_1

16

Delete Right

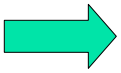
- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** *r* **from** $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x,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

17

Delete Right

- Precondition: $p \in S$, $y \in O$
- Primitive command: **delete** *r* **from** $a[p, y]$

	p	q
x	r_1	r_1,r_2
y	r	r_1



	p	q
x	r_1	r_1,r_2
y		r_1

18

Destroy Subject

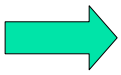
- Precondition: $s \in S$
- Primitive command: **destroy subject** s
- Postconditions:
  - $S' = S - \{ s \}$, $O' = O - \{ s \}$
  - ~~$(\forall y \in O')[a'[s, y] = \varnothing]$~~,
  - ~~$(\forall x \in S')[a'[x, s] = \varnothing]$~~
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

19

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** s

	p	q
x	r_1	r_1,
y		r_1
s	r_2	



	p	q
x	r_1	r_1,r_2
y		r_1

20

Destroy Object

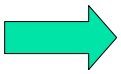
- Precondition: $o \in O$
- Primitive command: **destroy object** o
- Postconditions:
  - $S' = S,\ O' = O - \{ o \}$
  - ~~$(\forall x \in S')[a'[x, o] = \varnothing]$~~
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

21

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** o

	p	q	**o**
x	r_1	r_1,r_2	r_2
y		r_1	



	p	q
x	r_1	r_1,r_2
y		r_1

22

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
```

23

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

24

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

25

Multiple Conditions

- Let *p* give *q* the rights *r* and *w* 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
```

NOTE: `or` and `not` are not allowed in conditions.

26

Copy Right

- Allows possessor to give rights to another (sometimes called grant right)
- Copier may or may not surrender the right, depending on system
- 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

27

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
    - Cannot give rights to specific (set of) users
    - Cannot pass copy flag to specific (set of) users

28

Attenuation of Privilege

- Principle says you cannot 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.

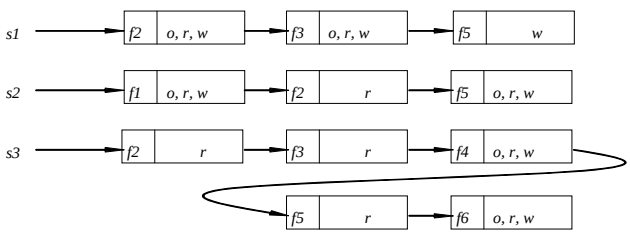
29

ACM implementation 1

o: own
r: read
w:write

	f1	*f2*	*f3*	*f4*	*f5*	*f6*
s1		*o, r, w*	*o, r, w*		*w*	
s2	*o, r, w*	*r*			*o, r, w*	
s3		*r*	*r*	*o, r, w*	*r*	*o, r, w*

Access Matrix



Capabilities

30

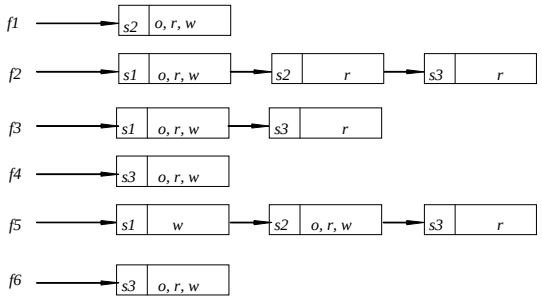
ACM implementation 2

o: own
r: read
w:write

	f1	f2	f3	f4	f5	f6
s1		o, r, w	o, r, w		w	
s2	o, r, w	r			o, r, w	
s3		r	r	o, r, w	r	o, r, w

Access Matrix

Access Control List



31

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

32