590J Lecture 21: Access Control (contd)

Review

- Recall:
  - Protection system is a description of conditions under which a system is secure
  - P is the set of all protection states
  - Q is the set of authorized protection states
    - Q secure system
    - P-Q insecure system
  - Secure policies characterize the states of Q
  - Security mechanisms ensure the system never enters P-Q

Review (contd)

- Access control matrix (A) relates
  - Objects (O) entities relevant to the protection state
  - Subjects (S) are active object
  - Rights (R) a subject has over an object; implementation dependent
- Example:

	file_1	file_2	proc_1
proc_1	*r,w,x*	*r*	*r,w,x,own*
proc_2	*r*	*r,w*	*r*

Protection State Transitions

- Process execution causes the protection system states to change:

$$t_{i+1} : X_i \rightarrow X_{i+1}$$

- This implies the access control matrix representation must change via *commands*:

$$c_{i+1}(p_{i+1,1},\ldots,p_{i+1,m}): X_i \rightarrow X_{i+1}$$

Primitive Commands

- Harrison, Ruzzo, and Ullman define a set of six *primitive commands* that alter the ACM:
  1. **create subject *s***
  2. **create object *o***
  3. **enter** *r* **into** *a*[*s,o*]
  4. **delete** *r* **from** *a*[*s,o*]
  5. **destroy subject** *s*
  6. **destroy object** *o*
- These primitive commands are used to construct more sophisticated commands
- Recall that $S \subseteq O$.

create subject *s*

- Precondition: $s \notin S$
- Postconditions: $S' = S \cup \{s\}$, $O' = O \cup \{s\}$, $(\forall y \in O')[a'[s,y] = \{\}]$, $(\forall x \in S')[a'[x,s] = \{\}]$, $(\forall x \in S)(\forall y \in O)[a'[x,y]=a[x,y]]$
- This primitive creates a new subject *s*, which must not exist as an object before command execution. Note that no rights are added to the matrix.

create object *o*

- Precondition: $o \notin O$
- Postconditions: $S' = S$, $O' = O \cup \{s\}$,
  $(\forall x \in S')[a'[x,o] = \{\}]$,
  $(\forall x \in S)(\forall y \in O)[a'[x,y]=a[x,y]]$
- This primitive creates a new object *o*, which must not exist as an object before command execution. Note that no rights are added to the matrix.

enter *r* **into** *a*[*s*,*o*]

- Precondition: $s \in S$, $o \in O$
- Postconditions: $S' = S$, $O' = O$, $a'[s,o] = a[s,o] \cup \{r\}$, $(\forall x \in S')(\forall y \in O')[(x,y) \neq (s,o) \rightarrow a'[x,y] = a[x,y]]$
- This command adds *r* to the set of rights at $a[s,o]$. If $r \in a[s,o]$ prior to the execution of the command, the behavior depends on the model instantiation.

delete *r* from *a*[*s*,*o*]

- Precondition: $s \in S$, $o \in O$
- Postconditions: $S' = S$, $O' = O$, $a'[s,o] = a[s,o] - \{r\}$, $(\forall x \in S')(\forall y \in O')[(x,y) \neq (s,o) \rightarrow a'[x,y] = a[x,y]]$
- This command removes *r* from the set of rights at $a[s,o]$. If $r \notin a[s,o]$ prior to the execution of the command, then the effect of the operation is null.

destroy subject *s*

- Precondition: $s \in S$
- Postconditions: $S' = S - \{s\}$, $O' = O - \{s\}$, $(\forall y \in O')[a'[s,y] = \varnothing]$, $(\forall x \in S')[a'[x,s] = \varnothing]$, $(\forall x \in S')(\forall y \in O')[a'[x,y]=a[x,y]]$
- This primitive deletes the subject *s* and the column/row defined by *s* in *A*.

destroy object *o*

- Precondition: $o \in O$
- Postconditions: $S' = S$, $O' = O - \{s\}$, $(\forall x \in S')[a'[x,o] = \varnothing]$, $(\forall x \in S')(\forall y \in O')[a'[x,y]=a[x,y]]$
- This primitive deletes the object *o* and removes the column defined by *o* from the matrix *A*.

Example: UNIX files

- Suppose a process *p* creates a file *f* with read and write permissions. Then *A* is updated with the following command:

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

Example: UNIX process

- Support a process *p* spawns a child process *q*. The following command updates the matrix *A*:

command *spawn-process* (*p*,*q*)
    create subject *q*;
    enter *own* **into** *a*[*p*,*q*];
    enter *r* **into** *a*[*p*,*q*];
    enter *w* **into** *a*[*p*,*q*];
    enter *r* **into** *a*[*q*,*p*];
    enter *w* **into** *a*[*q*,*p*];
end

interprocess signals

Example: Uni-operational commands

- Primitive commands are not meant to be used directly. Instead, a wrapper around them provides their functionality:

  command *make-owner* (*p*,*f*)
      enter *own* **into** *a*[*p*,*f*];
  end

Conditional Commands

- What if a process *p* wanted to give permission to read a file *f* to another process *q*?
- Process *p* would have to have the rights to that file.
  - Principle of Attenuation of Privilege: A subject s_1 may not grant rights to another subject s_2 of an object *o* that it does not have those rights to.
- Conditional statements in commands allow specific preconditions to be satisfied.

Conditional Commands (contd)

- Example: conjunction

command *grant-read-file* (*p*,*f*,*q*)
  if r **in** *a*[*p*,*f*] **and** *c* **in** *a*[*p*,*f*] **then**
    enter *r* **into** *a*[*q*,*f*];
end

- Disjunctions and negations are not allowed.
  - '**or**' can be represented as two commands
  - absence of rights is not permitted.

The *own* Right

- The *own* right allows
  - a subject to grant rights to other (may be restricted)
  - self-referential right granting
- The owner is usually the creator of an object
- Semantics get tricky:
  - Can new owners delete objects?
  - Should ownership be transferred?
  - Who is reponsible for the object?