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

objects (entities)

- 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 $r, w, x, a, o$

<table>
<thead>
<tr>
<th></th>
<th>$f$</th>
<th>$g$</th>
<th>$p$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>rwo</td>
<td>$r$</td>
<td>rwxo</td>
<td>$w$</td>
</tr>
<tr>
<td>$q$</td>
<td>$a$</td>
<td>ro</td>
<td>$r$</td>
<td>rwxo</td>
</tr>
</tbody>
</table>
Example 2

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

<table>
<thead>
<tr>
<th></th>
<th>counter</th>
<th>inc_ctr</th>
<th>dec_ctr</th>
<th>manage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inc_ctr</code></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>dec_ctr</code></td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>manage</code></td>
<td></td>
<td><code>call</code></td>
<td><code>call</code></td>
<td><code>call</code></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>age</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>teacher</td>
<td>45</td>
<td>$40,000</td>
</tr>
<tr>
<td>Bob</td>
<td>aide</td>
<td>20</td>
<td>$20,000</td>
</tr>
<tr>
<td>Cathy</td>
<td>principal</td>
<td>37</td>
<td>$60,000</td>
</tr>
<tr>
<td>Dilbert</td>
<td>teacher</td>
<td>50</td>
<td>$50,000</td>
</tr>
<tr>
<td>Eve</td>
<td>teacher</td>
<td>33</td>
<td>$50,000</td>
</tr>
</tbody>
</table>

Queries:
1. $\text{sum(salary, \text{"position = teacher"})} = $140,000
2. $\text{sum(salary, \text{"age > 40 & position = teacher"})}$ should not be answered (deduce Eve’s salary)

ACM of Database Queries

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

$f(o) = \{ \text{read} \}$ for $o \in O_i$, if $|\bigcup_{j=1,\ldots,i} O_j| \neq 2$, $n-1$

$f(o) = \emptyset$ for $o \in O_i$, otherwise

1. $O_1 = \{ \text{Alice, Dilbert, Eve} \}$, so $|O_1| = n$, and:
   - $A[\text{asker, Alice}] = f(\text{Alice}) = \{ \text{read} \}$
   - $A[\text{asker, Dilbert}] = f(\text{Dilbert}) = \{ \text{read} \}$
   - $A[\text{asker, Eve}] = f(\text{Eve}) = \{ \text{read} \}$

and query can be answered
But Query 2

From last slide:

\[ f(o_j) = \{ \text{read} \} \quad \text{for } o_j \in O_i, \text{ if } | \bigcup_{j=1}^n O_j | = 2, \ n-1 \]
\[ f(o_j) = \emptyset \quad \text{for } o_j \in O_i, \text{ otherwise} \]

2. \( O_2 = \{ \text{Alice, Dilbert} \} \) but \( |O_1 \cup O_2| = n-1 \), so

\[ A[\text{asker, Alice}] = f(\text{Alice}) = \emptyset \]
\[ A[\text{asker, Dilbert}] = f(\text{Dilbert}) = \emptyset \]

and query cannot be answered

State Transitions

- Change the protection state of system
- \( \rightarrow \) represents transition
  - \( X_i \rightarrow_\tau X_{i+1} \): command \( \tau \) moves system from state \( X_i \) to \( X_{i+1} \)
  - \( X_i \rightarrow_\ast X_{i+1} \): 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:**
  - \(S' = S \cup \{s\}\), \(O' = O \cup \{s\}\)
  - \((\forall y \in O')[a'[s, y] = \emptyset]\), \((\forall x \in S')[a'[x, s] = \emptyset]\)
  - \((\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]\)
Create Object

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

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: \textbf{delete} \( r \) \textbf{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: \textbf{destroy subject} \( s \)
• Postconditions:
  – \( S' = S - \{ s \}, \ O' = O - \{ s \} \)
  – \( (\forall y \in O')[a'[s, y] = \emptyset], \ (\forall x \in S')[a'[x, s] = \emptyset] \)
  – \( (\forall x \in S')(\forall y \in O') [a'[x, y] = a[x, y]] \)
Destroy Object

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

Creating File

• Process \( p \) creates file \( f \) with \( r \) and \( w \) permission

\begin{verbatim}
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
\end{verbatim}
Mono-Operational Commands

• Make process \( p \) the owner of file \( g \)

```plaintext
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 \( f \)

```plaintext
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 $q$

\[
\text{command grant}\cdot\text{read}\cdot\text{file}\cdot2(p, f, q) \text{ if own in } A[p, f] \text{ and } c \text{ in } A[p, q] \text{ then}
\]

\[
\begin{align*}
\text{enter } & r \text{ into } A[q, f]; \\
\text{enter } & w \text{ into } A[q, f]; \\
\text{end}
\end{align*}
\]

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 \textit{not} executed

• Body
  – May contain commands and/or primitive operations
  – May \textit{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 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 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