Lecture 2: Access Control Matrix

January 6, 2011
1. Modeling

2. What is an access control matrix?

3. Some examples
   - Boolean expressions for database control
   - History for program execution control

4. Formal model
   - Primitive operations
   - Types of commands

5. Propagating rights
   - Copy and own
   - Attenuation of privilege

6. What Next?
Models

- Abstract irrelevant details of entity or process being modeled
  - Allows you to focus on aspects that are of interest
  - *If done correctly*, results from analyzing the model apply to entity or process
- Assumption: nothing you omit affects the application of the results
Protection State

Protection state of system describes current settings, values relevant to protection

- Access control matrix representation of protection state
  - Describes protection state precisely
  - Matrix describing rights of subjects (rows) over objects (columns)
  - State transitions change elements of matrix

- **Subject** is active entities (processes, users, *etc.*)

- **Object** has 2 meanings:
  - Passive entity (*not* a subject)
  - Any entity acting passively (so can be a subject)

Context tells you which sense is used
What is an access control matrix?

- 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 $o_j$
Access Control Matrix for System

- Processes $p$, $q$
- Files $f$, $g$
- Rights $r$, $w$, $x$, $a$, $o$
  - Rights are merely symbols; interpretation depends on system
  - Example: on UNIX, $r$ means “read” for file and “list” for directory

```
+-----+-----+-----+-----+
|     | $f$  | $g$  | $p$  | $q$  |
|-----+-----+-----+-----+-----|
| $p$  | rwo | $r$  | rwxo | $w$  |
| $q$  | $a$  | ro   | $r$  | rwxo |
```
Access Control Matrix for Program

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights +, −, x, *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><em>inc_ctr</em></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>dec_ctr</em></td>
<td>−</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>manage</em></td>
<td></td>
<td><em>call</em></td>
<td><em>call</em></td>
<td><em>call</em></td>
</tr>
</tbody>
</table>
Access Control Matrix for Database

- Access control matrix shows allowed 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
  - Result controls granting, denying access
Boolean Expressions and Access

- Subject *annie*: attributes role (artist), groups (creative)
- Verb *paint*: default 0 (deny unless explicitly granted)
- Object *picture*: Rule is

\[
paint: \quad \text{‘artist’ in subject.role and}
\]
\[
\text{‘creative’ in subject.groups and}
\]
\[
time\text{.hour} \geq 0 \text{ and time\text{.hour}} < 5
\]
Example: ACM at 3 a.m. and 10 a.m.

At 3 a.m., time condition met; ACM is:

At 10 a.m., time condition not met; ACM is
Executing Downloaded Programs

- Downloaded programs may access system in unauthorized ways
  - Example: Download Trojan horse that modifies configuration, control files
- Condition access rights upon the rights of previously executed code (i.e., history)
  - Each piece of code has set of static rights
  - Executing process has set of current rights
  - When piece of code runs, its rights are set of current rights \( \cap \) set of static rights
Example Programs

main runs, loads helper_proc and runs it

// This routine has no filesystem access rights
// beyond those in a limited, temporary area
procedure helper_proc()
    return sys_kernel_file;
// But this has the right to delete files
program main()
    sys_load_file(helper_proc);
    file = helper_proc();
    sys_delete_file(file);

sys_kernel_file is system kernel
tmp_file file in limited, temporary area helper_proc can access
## History for program execution control

### Initial static rights:

<table>
<thead>
<tr>
<th></th>
<th><code>sys_kernel_file</code></th>
<th><code>tmp_file</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main</code></td>
<td>delete</td>
<td>delete</td>
</tr>
<tr>
<td><code>helper_proc</code></td>
<td></td>
<td>delete</td>
</tr>
</tbody>
</table>

### Program starts; its rights are those of `main`:

<table>
<thead>
<tr>
<th></th>
<th><code>sys_kernel_file</code></th>
<th><code>tmp_file</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main</code></td>
<td>delete</td>
<td>delete</td>
</tr>
<tr>
<td><code>helper_proc</code></td>
<td></td>
<td>delete</td>
</tr>
<tr>
<td><code>process</code></td>
<td>delete</td>
<td>delete</td>
</tr>
</tbody>
</table>

### After `helper_proc` called, process loses right to delete kernel:

<table>
<thead>
<tr>
<th></th>
<th><code>sys_kernel_file</code></th>
<th><code>tmp_file</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main</code></td>
<td>delete</td>
<td>delete</td>
</tr>
<tr>
<td><code>helper_proc</code></td>
<td></td>
<td>delete</td>
</tr>
<tr>
<td><code>process</code></td>
<td>delete</td>
<td>delete</td>
</tr>
</tbody>
</table>
State Transitions

- Represent changes to the protection state of the system
- \( \vdash \) represents transition
  - \( X_i \vdash_\tau X_{i+1} \): command \( \tau \) moves system from state \( X_i \) to state \( X_{i+1} \)
  - \( X_i \vdash^* X_{i+1} \): a sequence of commands moves system from state \( X_i \) to state \( X_{i+1} \)
- Commands sometimes 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:
  - \( S' = S, \ O' = O \cup \{ o \} \)
  - \( (\forall x \in S')[A'[x, o] = \emptyset] \)
  - \( (\forall x \in S)(\forall y \in O)[A'[x, y] = A[x, y]] \)
Primitive operations

enter

- **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

- **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:**
  - $S' = S - \{s\}$, $O' = O - \{s\}$
  - $(\forall y \in O')[A'[s, y] = \emptyset], (\forall x \in S')[A'[x, s] = \emptyset]$
  - $(\forall x \in S')(\forall y \in O')[A'[x, y] = A[x, y]]$
destroy object

- **Precondition:** \( o \in O \)
- **Primitive command:** `destroy object s`
- **Postconditions:**
  - \( S' = S, \; O' = O - \{o\} \)
  - \( (\forall x \in S')[A'[x, o] = \emptyset] \)
  - \( (\forall x \in S)(\forall y \in O)[A'[x, y] = A[x, y]] \)
Example: Creating File

Process $p$ creates file $f$ with $r$ and $w$ permissions

```plaintext
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 $f$
  
  \[
  \text{command make\textbullet owner}(p, f) \\
  \text{enter own into } A[p, f]; \\
  \text{end}
  \]

- Single primitive operation in this command
  
  - So it's \textit{mono-operational}
Conditional Commands

- If $p$ owns $f$, let $p$ give $q$ $r$ rights over $f$

\[
\text{command grant\textbullet rights}(p, f, q) \\
\text{if own in } A[p, f] \\
\text{then} \\
\text{enter } r \text{ into } A[q, f] \\
\text{end}
\]

- Single condition in this command
  - So it’s *mono-conditional*
Multiple Conditions

- If $p$ has both $r$ and $c$ rights over $f$, let $p$ give $q$ $r$ and $w$ rights over $f$

\[
\text{command grant\textbullet read\textbullet if\textbullet r\textbullet and\textbullet c(p, f, q)}
\]

\[
\text{if } r \text{ in } A[p, f] \text{ and } c \text{ in } A[p, q]
\]

\[
\text{then}
\]

\[
\text{enter } r \text{ into } A[q, f]
\]

\[
\text{enter } w \text{ into } A[q, f]
\]

\[
\text{end}
\]

- Two conditions in this command
  - So it's \textit{bi-conditional}
“Or” Conditions

- If \( p \) has either \( r \) or \( c \) rights over \( f \), let \( p \) give \( q \) \( r \) and \( w \) rights over \( f \)
  - No “or” operator, so we write command for each possibility
  - Then execute them sequentially
  - Note: if multiple conditions hold, actions may be taken more than once (usually to no effect)
$r, c$ Commands

command grant\textbullet read\textbullet file\textbullet if\textbullet r($p, f, q$)

\begin{align*}
\text{if } r \text{ in } A[p, f] \\
\text{then} \\
\quad \text{enter } r \text{ into } A[q, f] \\
\quad \text{enter } w \text{ into } A[q, f]
\end{align*}

end

command grant\textbullet read\textbullet file\textbullet if\textbullet c($p, f, q$)

\begin{align*}
\text{if } c \text{ in } A[p, f] \\
\text{then} \\
\quad \text{enter } r \text{ into } A[q, f] \\
\quad \text{enter } w \text{ into } A[q, f]
\end{align*}

end
$r$ or $c$ Command

```
command grant•read•file•if•r•or•c(p, f, q)
  grant•read•file•if•r(p, f, q);
  grant•read•file•if•c(p, f, q);
end
```
Copy

- 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
  - \( rc \) or \( r:c \) is read right that can be copied
  - In this case, called a copy flag
- Is copy flag copied with copying the associated right?
  - Depends on rules of model, or instantiation of model
Copy and own

Own

- Usually allows possessor to change entries in ACM column
  - Owner of object can add, delete rights over that object for others
- What can be done is system (instantiation) dependent
  - Some disallow giving rights to specific (set of) users
  - Some disallow passing of copy flag to specific (set of) users
Principle of Attenuation of Privilege

- You increase your rights
- You cannot give rights that 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
Very simple model, but very powerful

Will use this to examine decidability of security

Will use very simple definition of “secure”:

- Adding a generic right $r$ where there was not one is leaking
- If a system $S$ begins in initial state $s_0$ and it cannot leak right $r$, we consider it secure with respect to the right $r$

We will formalize this and study it