ECS 235B Module 4
Access Control Matrix
### Description

- **Subjects**: $S = \{ s_1, \ldots, s_n \}$
- **Objects**: $O = \{ o_1, \ldots, o_m \}$
- **Rights**: $R = \{ r_1, \ldots, r_k \}$
- **Entries**: $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_{x_1}, \ldots, r_{y_1} \}$ means subject $s_i$ has rights $r_{x_1}, \ldots, r_{y_1}$ 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

• Host names *telegraph*, *nob*, *toadflax*

• Rights *own*, *ftp*, *nfs*, *mail*

<table>
<thead>
<tr>
<th></th>
<th>telegraph</th>
<th>nob</th>
<th>toadflax</th>
</tr>
</thead>
<tbody>
<tr>
<td>telegraph</td>
<td>own</td>
<td>ftp</td>
<td>ftp</td>
</tr>
<tr>
<td>nob</td>
<td>ftp, mail, nfs, own</td>
<td>ftp, nfs, mail</td>
<td></td>
</tr>
<tr>
<td>toadflax</td>
<td>ftp, mail</td>
<td>ftp, mail, nfs, own</td>
<td></td>
</tr>
</tbody>
</table>
Example 3

• Procedures $inc_{\text{ctr}}$, $dec_{\text{ctr}}$, $manage$
• Variable $counter$
• Rights $+, -, call$

<table>
<thead>
<tr>
<th></th>
<th>counter</th>
<th>$inc_{\text{ctr}}$</th>
<th>$dec_{\text{ctr}}$</th>
<th>$manage$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$inc_{\text{ctr}}$</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$dec_{\text{ctr}}$</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$manager$</td>
<td></td>
<td>$call$</td>
<td>$call$</td>
<td>$call$</td>
</tr>
</tbody>
</table>
UNIX/Linux Access Controls

• Files
  • A is ~bishop/a.out (0755, or rwxr-xr-x)
  • B is /etc/passwd (0644, or rw-r--r--)
  • H is /home/bishop (0711, or rwx-x-x-x)
  • S is /bin/su (4711, or s--rwx--x--x)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>bishop</td>
<td>rwxo</td>
<td>r</td>
<td>x</td>
<td>rwxo</td>
</tr>
<tr>
<td>zheng</td>
<td>rx</td>
<td>r</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>root</td>
<td>rwx</td>
<td>rwo</td>
<td>rwxo</td>
<td>rwx</td>
</tr>
</tbody>
</table>
UNIX/Linux Access Controls

• Access control matrices are dynamic:

• After bishop executes `chmod 700 /home/bishop`:
  • Same as `chmod u=rwx,g-rwx,o-rwx /home/bishop`

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>bishop</td>
<td>rwxo</td>
<td>r</td>
<td>x</td>
<td>rwxo</td>
</tr>
<tr>
<td>muwei</td>
<td>r</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>root</td>
<td>rwx</td>
<td>rwo</td>
<td>rwxo</td>
<td>rwx</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), *group* (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 ≤ 4`
ACM at 3AM and 10AM

At 3AM, time condition met
ACM is:

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

• Problem: what a process has accessed may affect what it can access now

• Example: procedure in a web applet can access other procedures depending on what procedures it has already accessed
  • $S$ set of *static rights* associated with procedure
  • $C$ set of *current rights* associated with each executing process
  • When process calls procedure, rights are $S \cap C$
Example Program

// 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)
    tmp_file = helper_proc()
    sys_delete_file(tmp_file)

• \textit{sys\_kernel\_file} contains system kernel

• \textit{tmp\_file} is in limited area that \textit{helper\_proc()} can access
Before \textit{helper\_proc} Called

- Static rights of program

\begin{tabular}{l|c|c}
\textbf{Function} & \textbf{sys\_kernel\_file} & \textbf{tmp\_file} \\
\hline
\texttt{main} & delete & delete \\
\texttt{helper\_proc} & & delete \\
\end{tabular}

- When program starts, current rights:

\begin{tabular}{l|c|c}
\textbf{Function} & \textbf{sys\_kernel\_file} & \textbf{tmp\_file} \\
\hline
\texttt{main} & delete & delete \\
\texttt{helper\_proc} & & delete \\
\texttt{process} & delete & delete \\
\end{tabular}
After *helper_proc* Called

- Process rights are intersection of static, previous “current” rights:

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

• Change the protection state of system
• $\vdash$ represents transition
  • $X_i \vdash_\tau X_{i+1}$: command $\tau$ moves system from state $X_i$ to $X_{i+1}$
  • $X_i \vdash ^* Y$: a sequence of commands moves system from state $X_i$ to $Y$
• 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: \( o \not\in O \)

• Primitive command: \texttt{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:
  • \( S' = S, \, O' = O \)
  • \( A'[s, \, o] = A[s, \, o] - \{ \, r \} \)
  • \((\forall x \in S')(\forall y \in O' - \{ \, o \}) \left[ A'[x, \, y] = A[x, \, y] \right] \)
  • \((\forall x \in S' - \{ \, s \})(\forall y \in O') \left[ A'[x, \, y] = A[x, \, y] \right] \)
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 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

```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 $g$

  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 $q$ rights over $f$, if $p$ owns $f$

  command grant\cdot read\cdot file\cdot 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 (and)

- Let $p$ give $q$ $r$ and $w$ rights over $f$, if $p$ owns $f$ and $p$ has $c$ rights over $q$
  
  \[
  \text{command } grant\cdot read\cdot file\cdot 2(p, f, q) \\
  \quad \text{if own in } A[p, f] \text{ and } c \text{ in } A[p, q] \\
  \quad \text{then} \\
  \quad \quad \text{enter } r \text{ into } A[q, f]; \\
  \quad \quad \text{enter } w \text{ into } A[q, f]; \\
  \quad \text{end}
  \]
There Is No “or”

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

  ```
  command grant·read·file·3($p$, $f$, $q$)
  if own in $A[p, f]$
  then
    enter $r$ into $A[q, f]$;
    enter $w$ into $A[q, f]$;
  end
  command grant·read·file·4($p$, $f$, $q$)
  if $c$ in $A[p, q]$
  then
    enter $r$ into $A[q, f]$;
    enter $w$ into $A[q, f]$;
  end
  grant·read·file·3($p$, $f$, $q$);
  grant·read·file·4($p$, $f$, $q$)
  ```
General Form

**command** name of command(parameters)

  if conditions (if many, separate with **and**)
  then

    list of commands or primitive operations to be executed;

end

• Only one **if**, and it must come **before** any primitive operations or subcommands

• When there is an **if**, no commands may follow it (but there can be commands in the body of the **if**)

• There is no **else**
Copy Flag and Right

• Allows possessor to give rights to another

• Often attached to a right (called a flag), 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 increase your rights, or 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.