Models: Access Control

• What is access control?
  – Limiting who is allowed to do what
• What is an access control model?
  – Specifying who is allowed to do what
• What makes this hard?
  – Interactions between types of access

See Harrison, Ruzzo, Ullman CACM'76
Basics

• State: Status of the system
  – Protection state: subset that deals with protection
• Access Control Matrix
  – Describes protection state
• Formally:
  – Objects $O$
  – Subjects $S$
  – Matrix $A \subseteq S \times O$
• Tuple $(S, O, A)$ defines protection states of system

Access Restriction Facility

• Subject: attributes (name, role, groups)
• Verbs: possible actions
  – Default rule for each verb
• Objects associated with set of verbs
  – Rule for each (object, verb) pair
  – Rule may be function of subject attributes
• Can be converted to Access Control Matrix
### Access Control Matrix:
#### Boolean Evaluation Example

<table>
<thead>
<tr>
<th></th>
<th>Internal</th>
<th>Local</th>
<th>State University</th>
<th>Long Distance</th>
<th>International</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>CR</td>
<td>T</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student</td>
<td>CR</td>
<td>T</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Staff</td>
<td>CR</td>
<td>T</td>
<td>R</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>Account</td>
<td>CR</td>
<td>T</td>
<td>CR</td>
<td>CR</td>
<td>CR</td>
</tr>
</tbody>
</table>

#### Description

- Subjects: $S = \{ s_1, \ldots, s_n \}$
- Objects: $O = \{ o_1, \ldots, o_m \}$
- Rights: $R = \{ r_1, \ldots, r_k \}$
- Entries: $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject $s_i$ has rights $r_x, \ldots, r_y$ over object $o_j$
Example 2

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights +, −, *call*

<table>
<thead>
<tr>
<th></th>
<th>inc_ctr</th>
<th>dec_ctr</th>
<th>manage</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>counter</em></td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>inc_ctr</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>dec_ctr</em></td>
<td>−</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>manage</em></td>
<td><em>call</em></td>
<td><em>call</em></td>
<td><em>call</em></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
    ```

Protection State Transitions

- **State** $X_i = (S_i, O_i, A_i)$
- **Transitions** $\tau_i$
  - Single transition $X_i \vdash_{\tau_{i+1}} X_{i+1}$
  - Series of transitions $X \vdash ^* Y$
- **Access control matrix may change**
  - Change command $c$ associated with transition
    - $X_i \vdash_{c_{i+1} (p_{j+1}, .., p_{j+1})} X_{i+1}$
- **Commands often called transformation procedures**
Special Privileges: Copy, Ownership

• Copy (or grant)
  – Possessor can extend privileges to another
• Own right
  – Possessor can change their own privileges
• Principle of Attenuation of Privilege
  – A subject may not give rights it does not possess

Primitive Commands
(Harrison, Ruzzo, Ullman CACM’76)

• Create Object o
  – Adds o to objects with no access
  – $S' = S$, $O' = O \cup \{o\}$, $(\forall x \in S')[a'[x, o] = \emptyset]$, $(\forall x \in S')(\forall y \in O)[a'[x, y] = a(x, y)]$
• Create Subject s
  – Adds s to objects, subjects, sets relevant access control to $\emptyset$
• Enter r into $a[s, o]$
• Delete r from $a[s, o]$
• Destroy subject s, destroy object o
Create Subject

- Precondition: \( s \notin S \)
- Primitive command: \texttt{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: \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:**
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x, y \in S \times O - \{ s, o \})$ \[ a'[x, y] = a[x, y] \]

Delete Right

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

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

Destroy Object

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

- Process $p$ creates file $f$ with $r$ and $w$ permission
  
  **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 $q\ r$ 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

Multiple Conditions

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

  command $grant\cdot readwrite\cdot file\cdot 2(p, f, q)$
  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
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

What is Secure?

• A secure system doesn’t allow violations of policy
  – Is this a good definition?
  – Can we use it?
• Alternative view: based on rights
  – Start with access control matrix $A$
  – $\text{Leak}$: commands can add right $r$ to an element of $A$ not containing $r$
  – $\text{Safe}$: System is safe with respect to $r$ if $r$ cannot be leaked
Formally:

• Given
  – initial state $X_0 = (S_0, O_0, A_0)$
  – Set of primitive commands $c$

• Can we reach a state $X_n$ where $\exists s,o$ such that $A_n[s,o]$ includes a right $r$ not in $A_0[s,o]$?
  – If so, the system is not safe
  – But is “safe” secure?

Are commands correctly implemented?

Example: Unix File System

• Access Control Matrix
  – Root has access to all files
  – Owner has access to their own files

• Safe with respect to file access right?
  – No chmod/chown command
  – Only “root” can get root privileges
  – Only user can authenticate as themselves

Is “Safe” definition useful?
Solution: Trust

- Safety doesn’t distinguish leak from authorized transfer of rights
- Subjects authorized to receive transfer of rights deemed “trusted”
  - Eliminate trusted subjects from matrix

Decidability Result

(Harrison, Ruzzo, Ullman CACM’76)

- Given a system where each command consists of a single primitive command, There exists an algorithm that will determine if a protection system with initial state $X_0$ is safe with respect to right $r$.
- Proof: determine minimum commands $k$ to leak
  - Delete/destroy: Can’t leak (or be detected)
  - Create/enter: new subjects/objects “equal”, so treat all new subjects as one
  - If $n$ rights, leak possible, must be able to leak in $n(|S_0|+1)(|O_0|+1)+1$ commands
- Enumerate all possible to decide
Decidability: Non-Primitive Commands

- It is undecidable if a given state of a given protection system is safe for a given generic right
- Proof: Reduction from halting problem
  - Symbols, states ⇒ rights
  - Tape cell ⇒ subject (can create new subjects)
  - Right own: \( s_i \) owns \( s_i+1 \) for \( 1 \leq i < k \)
  - Cell \( s_i \) A ⇒ \( s_i \) has A rights on itself
  - Cell \( s_k \) ⇒ \( s_k \) has end rights on itself
  - State \( p \), head at \( s_i \) ⇒ \( s_i \) has \( p \) rights on itself
CERIAS Student Orientation and Welcome

- Wednesday, 1 September, 5:45-7:15 p.m., LWSN 1142
- Faculty and projects will be introduced
- Learn about the
  - Official CERIAS Student Association
  - New funding opportunities
- CERIAS students are expected to attend.
  - Visitors, faculty and staff are welcome
- Photos will be taken
  - If you have a photo url available, e.g., Departmental photo, please send to walls@cerias.purdue.edu
- Pizza will be served
  - RSVP walls@cerias.purdue.edu with number of pizza slices you would like by 9 a.m. Wednesday, Sept 1

Example:

<table>
<thead>
<tr>
<th>Turing Machine</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
</tr>
<tr>
<td>s1</td>
<td>A</td>
</tr>
<tr>
<td>s2</td>
<td>B</td>
</tr>
<tr>
<td>s3</td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td></td>
</tr>
</tbody>
</table>

9/13/2010  CS52600  37

© 2010 Chris Clifton  18
After $\delta(k, C) = (k_1, X, R)$ where $k$ is the current state and $k_1$ the next state

\[
\begin{array}{c|c|c|c|c}
\text{s}_1 & \text{s}_2 & \text{s}_3 & \text{s}_4 \\
\hline
\text{s}_1 & \text{A} & \text{own} & & \\
\hline
\text{s}_2 & \text{B} & \text{own} & & \\
\hline
\text{s}_3 & & \text{X} & \text{own} & \\
\hline
\text{s}_4 & & & \text{D} & \text{k}_1 \text{ end} \\
\end{array}
\]
Commands:

- Halting problem Turing Machine: Symbols A, B; states p, q
- $C_{p,A}(s_i, s_{i-1})$ (move left)
  - if $own \in a[s_{i-1}, s_i]$ and $p \in a[s_i, s_i]$ and $A \in a[s_i, s_i]$
    - Delete p from $a[s_i, s_i]$, A from $a[s_i, s_i]$
    - Enter B into $a[s_i, s_i]$, q into $a[s_{i-1}, s_{i-1}]$
- Similar commands for move right, move right at end of tape
- Simulates Turing machine
  - Leaks halting state $\Rightarrow$ halting state in the matrix $\Rightarrow$ Halting state reached
  
  This is undecidable!

Mapping

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>X</td>
<td>Y</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

head

After $\delta(k_1, D) = (k_2, Y, R)$
where $k_1$ is the current state and $k_2$ the next state

<table>
<thead>
<tr>
<th></th>
<th>s_1</th>
<th>s_2</th>
<th>s_3</th>
<th>s_4</th>
<th>s_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_1</td>
<td>A</td>
<td>own</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_2</td>
<td>B</td>
<td>own</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_3</td>
<td>X</td>
<td>own</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_4</td>
<td></td>
<td>Y</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_5</td>
<td></td>
<td></td>
<td></td>
<td>b k_2</td>
<td>end</td>
</tr>
</tbody>
</table>
Command Mapping

\[ \delta(k_1, D) = (k_2, Y, R) \text{ at end becomes} \]

```plaintext
command crightmost_{k,C}(s_4, s_5)
if end in A[s_4, s_4] and k_1 in A[s_4, s_4]
   and D in A[s_4, s_4]
then
   delete end from A[s_4, s_4];
   create subject s_5;
   enter own into A[s_4, s_5];
   enter end into A[s_5, s_5];
   delete k_1 from A[s_4, s_4];
   delete D from A[s_4, s_4];
   enter Y into A[s_4, s_4];
   enter k_2 into A[s_5, s_5];
end
```

Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 end right in ACM
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If TM enters state \( q_f \), then right has leaked
- If safety question decidable, then represent TM as above and determine if \( q_f \) leaks
  - Implies halting problem decidable
- Conclusion: safety question undecidable
Other Results

*(most from the same authors)*

- Set of unsafe systems recursively enumerable
- Without create primitive, safety in P-SPACE
  - Like halting problem reduction, but no unlimited tape
- Without delete/destroy, still undecidable
  - Decidable if at most one condition allowed per command
  - Still holds if delete allowed

---

Mono-Operational Commands

- Answer: yes
- Sketch of proof:
  Consider minimal sequence of commands $c_1, \ldots, c_k$ to leak the right.
  - Can omit *delete*, *destroy*
  - Can merge all *creates* into one
Worst case: insert every right into every entry; with $s$ subjects and $o$ objects initially, and $n$ rights, upper bound is $k \leq n(s+1)(o+1)$
What Else Might We Add?

• Default Rule
  – General default: Receive
  – Object default: Call Internal
  – Requires ability to override with negative and positive access
• Time-based access
  – Allow students to call on State University system after hours?
• History-based access

Access Control by History

• Example: Statistical Database
  – Allows queries for general statistics
  – But not individual values
• Valid queries: Statistics on 20+ individuals
  – Total salary of all Deans
  – Salary of Computer Science Professors
• See a problem coming?
  – Salary of CS Professors who aren’t Deans
Solution: Query Set Overlap Control  
(Dobkin, Jones & Lipton ’79)

• Query valid if intersection of query coverage and each previous query < r
• Given K minimum query size, r overlap:
  – Need 1 + (K-1)/r queries to compromise
• Can represent as access control matrix
  – Subjects: entities issuing queries
  – Objects: Powerset of records
  – $O_s(i)$: objects referenced by $s$ in queries 1..i
  – $A[s,o] = \text{read iff } \forall q \in q_i \cup \bigcap_{j<i} q_j <$

Next

• Optional reading: Dobkin, Jones, and Lipton (TODS 4(1), see course web site)
• Basic theorems on protection states
  – Decidability of safety of a state with respect to a right
• More Protection Models
Protection Study: Your Homework

• What does it take to make sure your homework is secure?
  – Let’s assume a Unix system (mentor.ics)
  – Issues?
• Participation Expected!

Where does this leave us?

• Safety decidable for some models
  – Are they practical?
• Safety only works if maximum rights known in advance
  – Policy must specify all rights someone could get, not just what they have
  – Where might this make sense?
• Next: Example of a decidable model
  – Take-Grant Protection Model