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.
Foundational Results

- Overview
- Harrison-Ruzzo-Ullman result
  – Corollaries
- Take-Grant Protection Model
- SPM and successors
Overview

- Safety Question
- HRU Model
- Take-Grant Protection Model
- SPM, ESPM
  - Multiparent joint creation
- Expressive power
- Typed Access Matrix Model
What Is “Secure”? 

- Adding a generic right \( r \) where there was not one is “leaking”
- If a system \( S \), beginning in initial state \( s_0 \), cannot leak right \( r \), it is safe with respect to the right \( r \).
Safety Question

• Does there exist an algorithm for determining whether a protection system $S$ with initial state $s_0$ is safe with respect to a generic right $r$?
  – Here, “safe” = “secure” for an abstract model
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 $k \leq n(s+1)(o+1)$
General Case

- Answer: no
- Sketch of proof:
  Reduce halting problem to safety problem
  Turing Machine review:
  - Infinite tape in one direction
  - States $K$, symbols $M$; distinguished blank $b$
  - Transition function $\delta(k, m) = (k', m', L)$ means in state $k$, symbol $m$ on tape location replaced by symbol $m'$, head moves to left one square, and enters state $k'$
  - Halting state is $q_f$; TM halts when it enters this state
Mapping

Current state is $k$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>...</td>
</tr>
</tbody>
</table>

head

```
<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>A</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td>B</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td></td>
<td>C</td>
<td>$k$</td>
<td>own</td>
</tr>
<tr>
<td>$s_4$</td>
<td></td>
<td></td>
<td>D</td>
<td>end</td>
</tr>
</tbody>
</table>
```
Mapping

After $\delta(k, C) = (k_1, X, R)$ where $k$ is the current state and $k_1$ the next state
Command Mapping

\[ \delta(k, C) = (k_1, X, R) \] at intermediate becomes

\[
\text{command } c_{k,C}(s_3, s_4)
\]

\[
\text{if own in } A[s_3, s_4] \text{ and } k \text{ in } A[s_3, s_3]
\]

\[
\text{and } C \text{ in } A[s_3, s_3]
\]

\[
\text{then}
\]

\[
\text{delete } k \text{ from } A[s_3, s_3];
\]

\[
\text{delete } C \text{ from } A[s_3, s_3];
\]

\[
\text{enter } X \text{ into } A[s_3, s_3];
\]

\[
\text{enter } k_1 \text{ into } A[s_4, s_4];
\]

\[
\text{end}
\]
Mapping

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

\[
\begin{array}{|c|c|c|c|c|}
\hline
s_1 & s_2 & s_3 & s_4 & s_5 \\ 
\hline
s_1 & A & own & & & \\
\hline
s_2 & B & own & & & \\
\hline
s_3 & & X & own & & \\
\hline
s_4 & & Y & own & & \\
\hline
s_5 & & & & b k_2 end & \\
\hline
\end{array}
\]
Command Mapping

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

\[
\text{command crightmost}_{k,c}(s_4, s_5) \\
\text{if end in } A[s_4, s_4] \text{ and } k_1 \text{ in } A[s_4, s_4] \\
\quad \text{and } D \text{ in } A[s_4, s_4] \\
\text{then} \\
\quad \text{delete end from } A[s_4, s_4]; \\
\quad \text{create subject } s_5; \\
\quad \text{enter own into } A[s_4, s_5]; \\
\quad \text{enter end into } A[s_5, s_5]; \\
\quad \text{delete } k_1 \text{ from } A[s_4, s_4]; \\
\quad \text{delete } D \text{ from } A[s_4, s_4]; \\
\quad \text{enter } Y \text{ into } A[s_4, s_4]; \\
\quad \text{enter } k_2 \text{ into } A[s_5, s_5]; \\
\text{end}
\]
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

• Set of unsafe systems is recursively enumerable
• Delete `create` primitive; then safety question is complete in P-SPACE
• Delete `destroy`, `delete` primitives; then safety question is undecidable
  – Systems are monotonic
• Safety question for monoconditional, monotonic protection systems is decidable
• Safety question for monoconditional protection systems with `create`, `enter`, `delete` (and no `destroy`) is decidable.
Take-Grant Protection Model

- A specific (not generic) system
  - Set of rules for state transitions
- Safety decidable, and in time linear with the size of the system
- Goal: find conditions under which rights can be transferred from one entity to another in the system
System

- objects (files, …)
- subjects (users, processes, …)
- don't care (either a subject or an object)

G \vdash x G'
apply a rewriting rule x (witness) to G to get G'

G \vdash^* G'
apply a sequence of rewriting rules (witness) to G to get G'

R = \{ t, g, r, w, … \} set of rights
Rules

take

\[ \begin{align*}
& \text{take} & \text{grant} \\
& \alpha & \alpha \\
\end{align*} \]

\[ \begin{align*}
& \alpha \\
\end{align*} \]
More Rules

create \[\bullet \quad \vdash \quad \bullet \quad \alpha \rightarrow \times\]

remove \[\bullet \quad \alpha \rightarrow \times \quad \vdash \quad \bullet \quad \alpha - \beta \rightarrow \times\]

These four rules are called the *de jure* rules.
Symmetry

1. $x$ creates ($tg$ to new) $v$
2. $z$ takes ($g$ to $v$) from $x$
3. $z$ grants ($\alpha$ to $y$) to $v$
4. $x$ takes ($\alpha$ to $y$) from $v$

Similar result for grant
Islands

- $tg$-path: path of distinct vertices connected by edges labeled $t$ or $g$
  - Call them “$tg$-connected”
- island: maximal $tg$-connected subject-only subgraph
  - Any right one vertex has can be shared with any other vertex
Initial, Terminal Spans

- **initial span** from \( x \) to \( y \)
  - \( x \) subject
  - \( tg \)-path between \( x \), \( y \) with word in \( \{ t^g \} \cup \{ \nu \} \)
  - Means \( x \) can give rights it has to \( y \)

- **terminal span** from \( x \) to \( y \)
  - \( x \) subject
  - \( tg \)-path between \( x \), \( y \) with word in \( \{ t^* \} \cup \{ \nu \} \)
  - Means \( x \) can acquire any rights \( y \) has