Foundational Results

- Safety Question
- HRU Model
- Take-Grant Protection Model
- Expressive power
- Typed Access Matrix Model

What Is “Secure”?

- Giving a generic right $r$ to a subject who did not initially possessed it is called “leaking”
- If a system $S$, beginning in initial state $s_0$, cannot leak right $r$, it is safe with respect to the right $r$.
- Leaking a right is not inherently bad
  - Legitimate transfer of rights by owner
- 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 $r$?
Formally:

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

- Can we reach a state $X_n$ ($X_0 \vdash^* X_n$) where
  $\exists s \in S$ and $\exists o \in O$ 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 a “safe” system a secure system?
    Are commands correctly implemented?

Trust

- Safety does not distinguish a leak of a right from an authorized transfer of rights
- Subjects authorized to receive transfer of rights deemed “trusted”
  - Eliminate trusted subjects from matrix
- Trivial cases of safety
  - $r = \text{read, own} \in a[s,o]$, command $\text{can\grant\read\if\own}$
  - No command includes the $\text{enter}$ primitive command

How about the general case?
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 (since new subjects are all equal)
- 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
Current state is $k$

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

\[ \text{command } c_{k,C}(s_3, s_4) \]
\[ \text{if } \text{own in } A[s_3, s_4] \text{ and } k \text{ in } A[s_3, s_3] \]
\[ \text{and } C \text{ in } A[s_3, s_3] \]
\[ \text{then} \]
\[ \text{delete } k \text{ from } A[s_3, s_3]; \]
\[ \text{delete } C \text{ from } A[s_3, s_3]; \]
\[ \text{enter } X \text{ into } A[s_3, s_3]; \]
\[ \text{enter } k_1 \text{ into } A[s_4, s_4]; \]
\[ \text{end} \]

Mapping

<table>
<thead>
<tr>
<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>
</tr>
</tbody>
</table>

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>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>
</tr>
<tr>
<td>s_2</td>
<td>B</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_3</td>
<td>X</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_4</td>
<td></td>
<td>Y</td>
<td>own</td>
<td></td>
</tr>
<tr>
<td>s_5</td>
<td></td>
<td></td>
<td></td>
<td>b k_2 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_5];
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

- Set of unsafe systems is recursively enumerable
- Delete `create` primitive; then safety question is complete in **P-SPACE**
- Delete `destroy`, `delete` primitives (this system is called monotonic): safety question is undecidable
- Safety question for monoconditional, monotonic protection systems is decidable
- Safety question for monoconditional protection systems with `create`, `enter`, `delete` (and no `destroy`) is decidable.

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
- Can the safety of a particular system, with specific rules, be established?
Take-Grant Protection Model

- A specific (not generic) system
  - System represented as a directed graph
  - Set of graph rewriting rules for state transitions
- Safety is 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)

\[ G \xrightarrow{x} G' \] apply a rewriting rule \( x \) (witness) to \( G \) to get \( G' \)

\[ G \xrightarrow{*} G' \] apply a sequence of rewriting rules (witness) to \( G \) to get \( G' \)

\( R = \{ t, g, r, w, \ldots \} \) set of rights
Rules

- take
- grant

More Rules

- create
- remove

These four rules are called the *de jure* rules
Example: Shared Buffer

- Initially s has grant rights for processes p and q.
- s sets up a shared buffer for p,q with the following steps
  - s creates new object b
  - s grants \( \{r,w\} \) to b to p
  - s grants \( \{r,w\} \) to b to q

Symmetry

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

Example

![Diagram](attachment:image.png)
can\textbf{•}share Predicate

Definition:

\[\text{can•share}(r, x, y, G_0)\] if, and only if, there is a sequence of protection graphs \(G_0, \ldots, G_n\) such that \(G_0 \models^* G_n\) using only \textit{de jure} rules and in \(G_n\) there is an edge from \(x\) to \(y\) labeled \(r\).

can\textbf{•}share Properties

- If \(x\) and \(y\) are subjects in an island, then \(\text{can•share}(r, x, y, G_0)\)
  - Proof by induction using the properties of \(\text{tg}\)-connected subjects
- General result: \(\text{can•share}(r, x, y, G_0)\) is decidable using an algorithm of complexity \(O(|V| + |E|)\) where \(V\) and \(E\) are the vertices and edges in the graph
  - Proof omitted (Exercise)
Key Question

- Characterize class of models for which safety is decidable
  - Existence: Take-Grant Protection Model is a member of such a class
  - Universality: in general, the question undecidable, so for some models it is not decidable
- What is the dividing line?

Typed Access Matrix Model

- Like ACM, but with set of types $T$
  - All subjects, objects have types
  - Set of types for subjects $TS$
- Protection state is $(S, O, \tau, A)$
  - $\tau: O \rightarrow T$ specifies type of each object
  - If $X$ subject, $\tau(X)$ in $TS$
  - If $X$ object, $\tau(X)$ in $T - TS$
- Same rules as ACM except for create
Create Rules

- Subject creation
  - **create subject** *s* of type *ts*
  - *s* must not exist as subject or object when operation executed
  - *ts* ∈ *TS*

- Object creation
  - **create object** *o* of type *to*
  - *o* must not exist as object when operation executed
  - *to* ∈ *T* − *TS*

Create Subject

- Precondition: *s* ∉ *S*
- Primitive command: **create subject** *s* of type *t*
- Postconditions:
  - *S*’ = *S* ∪ { *s* }, *O*’ = *O* ∪ { *s* }
  - (∀ *y* ∈ *O*)[τ’(*y*) = τ(*y*), τ’(*s*) = *t*]
  - (∀ *y* ∈ *O*’)[a’[*s*, *y*] = Ø], (∀ *x* ∈ *S*’)[a’[*x*, *s*] = Ø]
  - (∀ *x* ∈ *S*)(∀ *y* ∈ *O*)[a’[*x*, *y*] = a[*x*, *y*]]
Create Object

- Precondition: \( o \notin O \)
- Primitive command: \textbf{create object} \( o \) \textbf{of type} \( t \)
- Postconditions:
  - \( S' = S, \ O' = O \cup \{o\} \)
  - \( (\forall y \in O)[\tau'(y) = \tau(y)], \ \tau'(o) = t \)
  - \( (\forall x \in S')[a'[x, o] = \emptyset] \)
  - \( (\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]] \)

Definitions

- MTAM (Monotonic TAM ) Model: TAM model without \textbf{delete, destroy}
- \( \alpha(x_1:t_1, ..., x_n:t_n) \) create command
  - \( t_i \) child type in \( \alpha \) if any of \textbf{create subject} \( x_i \) \textbf{of type} \( t_i \) or \textbf{create object} \( x_i \) \textbf{of type} \( t_i \) occur in body of \( \alpha \)
  - \( t_i \) parent type otherwise
Cyclic Creates

command havoc(s₁ : u , s₂ : u , o₁ : v , o₂ : v , o₃ : w , o₄ : w)
    create subject s₁ of type u;
    create object o₁ of type v;
    create object o₃ of type w;
    enter r into a[s₂, s₁] ;
    enter r into a[s₂, o₂] ;
    enter r into a[s₂, o₄] ;
End

What kind of types are u, v and w?

---

Creation Graph

- u, v, w child types
- u, v, w also parent types
- Graph: lines from parent types to child types
- This graph has cycles
Theorems

- Safety decidable for systems with acyclic MTAM schemes
- Safety for acyclic ternary MATM decidable in time polynomial in the size of the initial ACM
  - “ternary” means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM

Key Points

- Safety problem undecidable
- Limiting scope of systems can make problem decidable
- Types critical to safety problem’s analysis