Access Control Model

Foundational Results
Objective

- Understand the basic results of the HRU model
  - Safety issue
  - Turing machine
  - Undecidability
Protection System

- State of a system
  - Current values of
    - memory locations, registers, secondary storage, etc.
    - other system components

- Protection state (P)
  - A system state that is considered secure

- A protection system
  - Captures the conditions for state transition
  - Consists of two parts:
    - A set of generic rights
    - A set of commands
Protection System

- **Subject** ($S$: set of all subjects)
  - Eg.: users, processes, agents, etc.
- **Object** ($O$: set of all objects)
  - Eg.: Processes, files, devices
- **Right** ($R$: set of all rights)
  - An action/operation that a subject is allowed/disallowed on objects
  - Access Matrix $A$: $a[s, o] \subseteq R$
- **Set of Protection States**: $(S, O, A)$
  - Initial state $X_0 = (S_0, O_0, A_0)$
State Transitions

\(X_i \vdash_{\tau_{i+1}} X_{i+1}\) : upon transition \(\tau_{i+1}\), the system moves from state \(X_i\) to \(X_{i+1}\)

\(X \vdash^* Y\) : the system moves from state \(X\) to \(Y\) after a set of transitions

\(X_i \vdash c_{i+1} (p_{i+1,1}, p_{i+1,2}, \ldots, p_{i+1,m}) X_{i+1}\) : state transition upon a command

For every command there is a sequence of state transition operations
## Primitive commands (HRU)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create subject $s$</td>
<td>Creates new row, column in ACM; $s$ does not exist prior to this</td>
</tr>
<tr>
<td>Create object $o$</td>
<td>Creates new column in ACM; $o$ does not exist prior to this</td>
</tr>
<tr>
<td>Enter $r$ into $a[s, o]$</td>
<td>Adds $r$ right for subject $s$ over object $o$; Ineffective if $r$ is already there</td>
</tr>
<tr>
<td>Delete $r$ from $a[s, o]$</td>
<td>Removes $r$ right from subject $s$ over object $o$</td>
</tr>
<tr>
<td>Destroy subject $s$</td>
<td>Deletes row, column from ACM;</td>
</tr>
<tr>
<td>Destroy object $o$</td>
<td>Deletes column from ACM</td>
</tr>
</tbody>
</table>
Primitive commands (HRU)

Create subject $s$

- Creates new row, column in ACM;
- $s$ does not exist prior to this

Precondition: $s \notin S$

Postconditions:

\[ S' = S \cup \{ s \}, \quad O' = O \cup \{ s \} \]

\[
(\forall y \in O')[a'[s, y] = \emptyset] \quad \text{(row entries for s)}
\]

\[
(\forall x \in S')[a'[x, s] = \emptyset] \quad \text{(column entries for s)}
\]

\[
(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]
\]
Primitive commands (HRU)

Enter $r$ into $a[s, o]$ Adds $r$ right for subject $s$ over object $o$
Ineffective if $r$ is already there

Precondition: $s \in S, o \in O$
Postconditions:
$S' = S, O' = O$

$$a'[s, o] = a[s, o] \cup \{ r \}$$

$$(\forall x \in S')(\forall y \in O')$$

$$[(x, y) \neq (s, o) \rightarrow a'[x, y] = a[x, y]]$$
[Unix] process \( p \) creates file \( f \) with owner \textit{read} and \textit{write} \((r, w)\) will be represented by the following:

Command \textit{create\_file}(p, f)

Create object \( f \)

Enter \textit{own} into \( a[p,f] \)

Enter \textit{r} into \( a[p,f] \)

Enter \textit{w} into \( a[p,f] \)

End
Process p creates a new process q

Command `spawn_process(p, q)`

Create subject `q`;
Enter `own` into `a[p,q]`
Enter `r` into `a[p,q]`
Enter `w` into `a[p,q]`
Enter `r` into `a[q,p]`
Enter `w` into `a[q,p]`
End

Parent and child can signal each other
System commands

- Defined commands can be used to update ACM
  
  Command `make_owner(p, f)`
  
  Enter `own` into `a[p,f]`
  
  End

- Mono-operational:
  - the command invokes only one primitive
Conditional Commands

- Mono-operational + mono-conditional

Command `grant_read_file(p, f, q)`

If `own` in `a[p,f]`
Then
Enter `r` into `a[q,f]`
End
Conditional Commands

- Mono-operational + biconditional

Command $\text{grant\_read\_file}(p, f, q)$

If $r$ in $a[p,f]$ and $c$ in $a[p,f]$
Then
  Enter $r$ into $a[q,f]$
End

- Why not “OR”??
Fundamental questions

- How can we determine that a system is secure?
  - Need to define what we mean by a system being “secure”
- Is there a generic algorithm that allows us to determine whether a computer system is secure?
What is a secure system?

- A simple definition
  - A secure system doesn’t allow violations of a security policy
- Alternative view: based on distribution of rights
  - Leakage of rights: (unsafe with respect to right r)
    - Assume that $A$ representing a secure state does not contain a right $r$ in an element of $A$.

- A right $r$ is said to be leaked, if a sequence of operations/commands adds $r$ to an element of $A$, which did not contain $r$
What is a secure system?

- Safety of a system with initial protection state $X_0$
  - Safe with respect to $r$: System is *safe with respect to $r$* if $r$ can never be leaked
  - Else it is called *unsafe with respect to right $r$*. 
Safety Problem: 
formally

- Given
  - initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands $c$
  - $r$ is not in $A_0[s, o]$

- 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?
Undecidable Problems

- **Decidable Problem**
  - A decision problem can be solved by an algorithm that halts on all inputs in a finite number of steps.

- **Undecidable Problem**
  - A problem that cannot be solved for all cases by any algorithm whatsoever
Decidability Results
*(Harrison, Ruzzo, Ullman)*

- **Theorem:**
  - Given a system where each command consists of a single *primitive* command (mono-operational), there exists an algorithm that will determine if a protection system with initial state $X_0$ is safe with respect to right $r$. 
Decidability Results

(Harrison, Ruzzo, Ullman)

- 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
    - No test for absence
    - Tests on $A[s_1, o_1]$ and $A[s_2, o_2]$ have same result as the same tests on $A[s_1, o_1]$ and $A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2]$

- If $n$ rights leak possible, must be able to leak $k = n(|S_0|+1)(|O_0|+1)+1$ commands

- Enumerate all possible states to decide
Decidability Results

*(Harrison, Ruzzo, Ullman)*

- It is undecidable if a given state of a given protection system is safe for a given generic right
- For proof – need to know Turing machines and halting problem
The **halting problem**: Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts (the alternative is that it runs forever without halting).
Turing Machine & Safety problem

- **Theorem:**
  - It is undecidable if a given state of a given protection system is safe for a given generic right

- Reduce TM to Safety problem
  - If Safety problem is decidable then it implies that TM halts (for all inputs) – showing that the halting problem is decidable (contradiction)

- TM is an abstract model of computer
  - Alan Turing in 1936
Turing Machine

- TM consists of
  - A tape divided into cells; infinite in one direction
  - A set of tape symbols $M$
    - $M$ contains a special blank symbol $b$
  - A set of states $K$
  - A head that can read and write symbols
  - An action table that tells the machine how to transition
    - What symbol to write
    - How to move the head (‘L’ for left and ‘R’ for right)
    - What is the next state

Current state is $k$
Current symbol is $C$
Turing Machine

- Transition function $\delta(k, m) = (k', m', L)$:
  - in state $k$, symbol $m$ on tape location is replaced by symbol $m'$,
  - head moves to left one square, and TM enters state $k'$
- Halting state is $q_f$
  - TM halts when it enters this state

Current state is $k$
Current symbol is $C$
Let $\delta(k, C) = (k_1, X, R)$ where $k_1$ is the next state
Turing Machine

Let $\delta(k, C) = (k_1, X, R)$
where $k_1$ is the next state

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Current state is $k$
Current symbol is $C$

Let $\delta(k_1, D) = (k_2, Y, L)$
where $k_2$ is the next state

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

head

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
**TM2Safety Reduction**

Proof: Reduce TM to safety problem

- Symbols, States $\Rightarrow$ rights
- Tape cell $\Rightarrow$ subject
- Cell $s_i$ has $A$ $\Rightarrow$ $s_i$ has $A$ rights on itself
- Cell $s_k$ $\Rightarrow$ $s_k$ has end rights on itself
- State $p$, head at $s_i$ $\Rightarrow$ $s_i$ has $p$ rights on itself
- Distinguished Right $own$:
  - $s_i$ owns $s_{i+1}$ for $1 \leq i < k$

<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></td>
<td>C $k$</td>
<td>own</td>
</tr>
<tr>
<td>$s_4$</td>
<td></td>
<td></td>
<td></td>
<td>D end</td>
</tr>
</tbody>
</table>

Current state is $k$
Current symbol is $C$
Command Mapping
(Left move)

\[ \delta(k, C) = (k, X, L) \]

If head is not in leftmost command \( c_{k,C}(s_i, s_{i-1}) \)
if own in \( a[s_{i-1}, s_i] \) and \( k \) in \( a[s_i, s_i] \)
and \( C \) in \( a[s_i, s_i] \)
then
- delete \( k \) from \( A[s_i, s_i] \);
- delete \( C \) from \( A[s_i, s_i] \);
- enter \( X \) into \( A[s_i, s_i] \);
- enter \( k_1 \) into \( A[s_{i-1}, s_{i-1}] \);
End

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>( s_2 )</td>
<td>( B )</td>
<td>( own )</td>
<td></td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( C k )</td>
<td>( own )</td>
<td></td>
</tr>
<tr>
<td>( s_4 )</td>
<td>( D )</td>
<td>( end )</td>
<td></td>
</tr>
</tbody>
</table>
Command Mapping
(Left move)

\[ \delta(k, C) = (k_1, X, L) \]

**If head is not in leftmost command**

\[ c_{k,C}(s_i, s_{i-1}) \]

If own in \( a[s_{i-1}, s_i] \) and \( k \) in \( a[s_i, s_i] \) and \( C \) in \( a[s_i, s_i] \)
then
- delete \( k \) from \( A[s_i, s_i] \);
- delete \( C \) from \( A[s_i, s_i] \);
- enter \( X \) into \( A[s_i, s_i] \);
- enter \( k_1 \) into \( A[s_{i-1}, s_{i-1}] \);
End

If head is in leftmost both \( s_i, s_{i-1} \) are \( s_1 \)
Command Mapping (Right move)

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

**command** \( c_{k, C}(s_i, s_{i+1}) \)

if own in \( a[s_i, s_{i+1}] \) and \( k \) in \( a[s_i, s_i] \) and \( C \) in \( a[s_i, s_i] \) then

- delete \( k \) from \( A[s_i, s_i] \);
- delete \( C \) from \( A[s_i, s_i] \);
- enter \( X \) into \( A[s_i, s_i] \);
- enter \( k_1 \) into \( A[s_{i+1}, s_{i+1}] \);

end

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>( s_2 )</td>
<td>B</td>
<td>own</td>
<td></td>
</tr>
<tr>
<td>( s_3 )</td>
<td>C</td>
<td>( k )</td>
<td>own</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>D</td>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
**Command Mapping**

(Right move)

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

**command** \( c_{k, C}(s_i, s_{i+1}) \)

*if own in \( a[s_i, s_{i+1}] \) and \( k \) in \( a[s_i, s_i] \) and \( C \) in \( a[s_i, s_i] \) then*

- delete \( k \) from \( A[s_i, s_i] \);
- delete \( C \) from \( A[s_i, s_i] \);
- enter \( X \) into \( A[s_i, s_i] \);
- enter \( k_1 \) into \( A[s_{i+1}, s_{i+1}] \);

**end**

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>( s_2 )</td>
<td>B</td>
<td>own</td>
<td></td>
</tr>
<tr>
<td>( s_3 )</td>
<td>X</td>
<td>own</td>
<td></td>
</tr>
<tr>
<td>( s_4 )</td>
<td>D</td>
<td></td>
<td>k_1 end</td>
</tr>
</tbody>
</table>

Current state is \( k_1 \)

Current symbol is \( C \)

head
Command Mapping (Rightmost move)

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

command crightmost\(_{k_1,C}(s_i, s_{i+1})\)

if end in \(a[s_i, s_i]\) and \(k_1\) in \(a[s_i, s_i]\) and \(D\) in \(a[s_i, s_i]\)
then
delete end from \(a[s_i, s_i]\);
create subject \(s_{i+1}\);
enter own into \(a[s_i, s_{i+1}]\);
enter end into \(a[s_{i+1}, s_{i+1}]\);
delete \(k_1\) from \(a[s_i, s_i]\);
delete \(D\) from \(a[s_i, s_i]\);
enter \(Y\) into \(a[s_i, s_i]\);
enter \(k_2\) into \(A[s_i, s_i]\);
end

\[
\delta(k_1, C) = (k_2, Y, R)
\]
Command Mapping (Rightmost move)

Current state is \( k_1 \)

Current symbol is \( D \)

\[ \delta(k_1, D) = (k_2, Y, R) \]

\[ \delta(k_1, D) = (k_2, Y, R) \]

**command** crightmost\(_{k, C} (s_i, s_{i+1})\)

if end in \( a[s_i, s_i] \) and \( k_1 \) in \( a[s_i, s_i] \) and \( D \)
in \( a[s_i, s_i] \)

then

- delete end from \( a[s_i, s_i] \);
- create subject \( s_{i+1} \);
- enter own into \( a[s_i, s_{i+1}] \);
- enter end into \( a[s_{i+1}, s_{i+1}] \);
- delete \( k_1 \) from \( a[s_i, s_i] \);
- delete \( D \) from \( a[s_i, s_i] \);
- enter \( Y \) into \( a[s_i, s_i] \);
- enter \( k_2 \) into \( A[s_i, s_i] \);

end
Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 *end* right in ACM
  - Only 1 right corresponds to a state
  - Thus, at most 1 applicable command in each configuration of the TM

- 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
  - Leaks halting state $\implies$ halting state in the matrix $\implies$ Halting state reached

- Conclusion: safety question undecidable
Other results

- For protection system without the create primitives, (i.e., delete create primitive); the safety question is complete in \textsc{P-Space}

- It is undecidable whether a given configuration of a given monotonic protection system is safe for a given generic right
  - Delete \texttt{destroy, delete} primitives;
  - The system becomes monotonic as they only increase in size and complexity

- The safety question for biconditional monotonic protection systems is undecidable

- The safety question for monoconditional, monotonic protection systems is decidable

- The safety question for monoconditional protection systems with \texttt{create, enter, delete} (and no \texttt{destroy}) is decidable.