Lecture 3: Decidability

January 11, 2011
1. Review

2. Decidability of security
   - Mono-operational command case
   - General case

3. Take-Grant Protection Model
   - Sharing rights
   - Take-Grant Systems
   - Stealing rights
   - Conspiracy
Why no “or”? 

- Unnecessary!
- Break conditional expression into sequence of disjuncts
- Write command with same body for each disjunct
- Call them sequentially!
$r, c$ Commands

\[
\text{command } \textit{grant \cdot read \cdot file \cdotifr}(p, f) \\
\quad \text{if } r \text{ in } A[p, f] \\
\quad \text{then} \\
\quad \qquad \text{enter } r \text{ into } A[q, f]; \\
\quad \qquad \text{enter } w \text{ into } A[q, f]; \\
\quad \text{end}
\]

\[
\text{command } \textit{grant \cdot read \cdot file \cdot ifc}(p, f) \\
\quad \text{if } c \text{ in } A[p, f] \\
\quad \text{then} \\
\quad \qquad \text{enter } r \text{ into } A[q, f]; \\
\quad \qquad \text{enter } w \text{ into } A[q, f]; \\
\quad \text{end}
\]
$r$ or $c$ Command

```
command grant.read.file.ifrorc(p, f)
     grant.read.file.ifr(p, f)
     grant.read.file.ifc(p, f)
end
```
What is “Secure”? 

**Leaking**

Adding a generic right $r$ where there was not one is *leaking*.

**Safe**

If a system $S$, beginning in initial state $s_0$, cannot leak right $r$, it is *safe* with respect to the right $r$.

Here, “safe” = “secure” for an abstract model.
What is Does “Decidable” Mean?

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 \( r \)?
Mono-Operational Commands

Answer:
Yes!

Proof sketch:
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, $o$ objects, and $n$ rights initially, upper bound is $k \leq n(s + 1)(o + 1)$
Proof (1)

- Consider minimal sequences of commands (of length $m$) needed to leak $r$ from system with initial state $s_0$
  - Identify each command by the type of primitive operation it invokes
- Cannot test for absence of rights, so delete, destroy not relevant
  - Ignore them
- Reorder sequences of commands so all creates come first
  - Can be done because enters require subject, object to exist
- Commands after these creates check only for existence of right
Proof (2)

- It can be shown (see homework):
  - Suppose \( s_1, s_2 \) are created, and commands test rights in \( A[s_1, o_1], A[s_2, o_2] \)
  - Doing the same tests on \( A[s_1, o_1] \) and \( A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2] \) gives same result
  - Thus all *creates* unnecessary
    - Unless \( s_0 \) is empty; then you need to create it (1 *create*)

- In \( s_0 \):
  - \( |S_0| \) number of subjects, \( |O_0| \) number of objects, \( n \) number of (generic) rights

- In worst case, 1 create
  - So a total of at most \( (|S_0| + 1)(|O_0| + 1) \) elements

- So \( m \leq n(|S_0| + 1)(|O_0| + 1) \)
General Case

Answer:

No

Proof sketch:

1. Show arbitrary Turing machine can be reduced to safety problem
2. Then deciding safety problem means deciding the halting problem
Turing Machine Review

- Infinite tape in one direction
- States $K$, symbols $M$, distinguished blank $\mathcal{B}$
- State transition function $\delta(k, m) = (k', m', L)$ in state $k$ with symbol $m$ under the TM head replace $m$ with $m'$, move head left one square, enter state $k'$
- Halting state is $q_f$
Mapping

Turing machine with head over square 3 on tape, in state $k$ and its representation as an access control matrix:

- $o$ is own right
- $e$ is end right
### Mapping

<table>
<thead>
<tr>
<th>Turing machine</th>
<th>Access control matrix representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 ...</td>
<td>$s_1$ $s_2$ $s_3$ $s_4$ ...</td>
</tr>
<tr>
<td>A B X D ...</td>
<td>$s_1$ $o$</td>
</tr>
</tbody>
</table>

After $\delta(k, C) = (k_1, X, R)$, where $k$ is the previous state and $k_1$ the current state.
General case

Command Mapping

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

\[ \text{command } c_{k,C}(s_i, s_{i+1}) \]
\[ \text{if } o \text{ in } A[s_i, s_{i+1}] \text{ and } k \text{ in } A[s_i, s_i] \text{ and } C \text{ in } A[s_i, s_i] \]
\[ \text{then} \]
\[ \text{delete } k \text{ from } A[s_i, s_i]; \]
\[ \text{delete } C \text{ from } A[s_i, s_i]; \]
\[ \text{enter } X \text{ into } A[s_i, s_i]; \]
\[ \text{enter } k_1 \text{ into } A[s_{i+1}, s_{i+1}]; \]
\[ \text{end} \]
Mapping

Turing machine

<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>o</td>
</tr>
</tbody>
</table>

⇒

Access control matrix representation

<table>
<thead>
<tr>
<th></th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
<th>s₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>A</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₂</td>
<td>B</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₃</td>
<td>X</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₄</td>
<td></td>
<td></td>
<td>Y</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>s₅</td>
<td></td>
<td></td>
<td></td>
<td>k₂</td>
<td>e</td>
</tr>
</tbody>
</table>

After \( \delta(k₁, D) = (k₂, Y, R) \), where \( k₁ \) is the previous state and \( k₂ \) the current state
General case

Command Mapping

\[ \delta(k_1, D) = (k_2, Y, R) \] at intermediate becomes:

\texttt{command} \quad \texttt{crightmost}_{k,D}(s_i , s_{i+1})
\[ \text{if } e \text{ in } A[s_i , s_i] \text{ and } k_1 \text{ in } A[s_i , s_i] \text{ and } D \text{ in } A[s_i , s_i] \text{ then} \]
\[ \text{delete } e \text{ from } A[s_i , s_i]; \]
\[ \text{create subject } y; \]
\[ \text{enter } o \text{ into } A[s_i , s_{i+1}]; \]
\[ \text{enter } e \text{ into } A[s_{i+1} , s_{i+1}]; \]
\[ \text{delete } k_1 \text{ from } A[s_i , s_i]; \]
\[ \text{delete } D \text{ from } A[s_i , s_i]; \]
\[ \text{enter } Y \text{ into } A[s_i , s_i]; \]
\[ \text{enter } k_2 \text{ into } A[s_{i+1} , s_{i+1}]; \]
\texttt{end}
Rest of Proof

- Protection system exactly simulates a Turing machine
  - Exactly 1 end (e) right in access control matrix
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command

- If Turing machine enters state $q_f$, then right has leaked

- If safety question decidable, then represent TM as protection system and determine if $q_f$ leaks
  - This implies halting problem is decidable

- Conclusion: safety question undecidable
Other Results

- Set of unsafe symbols is recursively enumerable.
- Delete `create` primitive; then safety question is complete in P-SPACE.
- Delete `destroy`, `delete` primitives; then safety question is undecidable.
  - Such systems are called `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 (passive entities like files, ...)
- Subjects (active entities like users, processes ...)
- Don’t care (either a subject or an object)

\[ G \vdash_{x} G' \]

Apply rewriting rule \( x \) (witness) to \( G \) to get \( G' \)

\[ G \vdash^{*} G' \]

Apply a sequence of rewriting rules (witness) to \( G \) to get \( G' \)

\[ R = \{ t, g, \ldots \} \]

Set of rights
Take, Grant Rules

In these rules, $\beta \subseteq \alpha \subseteq R$

**Take Rule**

$x \cdot y \otimes z \otimes t \alpha \triangleright x \cdot y \otimes z \otimes g \beta$

**Grant Rule**

$x \otimes y \cdot z \otimes g \alpha \triangleright x \otimes y \cdot z \otimes t \beta$
Create, Remove Rules

create rule

\[
\begin{align*}
\text{create rule} & \quad \bullet \quad \vdash \quad \bullet \\
x & \quad \alpha & \quad t & \quad y
\end{align*}
\]

remove rule

\[
\begin{align*}
\text{remove rule} & \quad \bullet \quad \vdash \quad \bullet \\
x & \quad \alpha & \quad \alpha - \beta & \quad y
\end{align*}
\]

These four rules are the *de jure* rules
Symmetry of Take and Grant

\[
\begin{align*}
\text{Take} & : x \rightarrow y \rightarrow z \\
\text{Grant} & : y \rightarrow x \rightarrow z
\end{align*}
\]
Symmetry of Take and Grant

1. x creates \((tg\ to\ new)\ v\)
Symmetry of Take and Grant

1. $x$ creates \((tg \text{ to new}) \ v\)
2. $x$ grants \((g \text{ to } v) \text{ to } y\)
Symmetry of Take and Grant

1. \( x \) creates \( (tg \text{ to new}) \) \( v \)
2. \( x \) grants \( (g \text{ to } v) \) to \( y \)
3. \( y \) grants \( (\beta \text{ to } z) \) to \( v \)
Symmetry of Take and Grant

1. x creates \((tg \text{ to new}) \, v\)
2. x takes \((g \text{ to } v)\) from x
3. y grants \((\beta \text{ to } z)\) to v
4. x takes \((\beta \text{ to } z)\) from v
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 that a vertex in the island has, can be shared with any other vertex in the island
Initial, Terminal Spans

- **Initial span** from $x$ to $y$: $x$ can give rights it has to $y$
  - $x$subject
  - $tg$-path between $x$, $y$ with word in $\{ \rightarrow t^* \rightarrow g \} \cup \{ \nu \}$

- **Terminal span** from $x$ to $y$: $x$ can get rights $y$ has
  - $x$subject
  - $tg$-path between $x$, $y$ with word in $\{ \rightarrow t^* \} \cup \{ \nu \}$
Bridges

- bridge \textit{tg}-path between subjects \(x, y\), with associated word in 
  \(\{\overrightarrow{t^*}, \overleftarrow{t^*}, \overrightarrow{g \ t^*}, \overrightarrow{t^* \ g \ t^*}\}\)
  
  - rights can be transferred between the two endpoints
  - \textit{not} an island as intermediate vertices are objects
Example

- islands: \{p, u\}, \{w\}, \{y, s'\}
- bridges: u, v, w; w, x, y
- initial span: p (associated word \(\nu\))
- terminal span: s's (associated word \(\vec{t}\))
can·share Predicate

can·share(r, x, y, G₀) holds if, and only if, there is a sequence of protection graphs G₀, . . . , Gₙ such that G₀ ⊢* Gₙ using only de jure rules and in Gₙ there is an edge from x to y labeled r
Sharing rights

**can·share** Theorem

\[ \text{can·share}(r, x, y, G_0) \] holds if, and only if, there is an edge from \( x \) to \( y \) labeled \( r \) in \( G_0 \), or the following hold simultaneously:

- there is an \( s \) in \( G_0 \) with an \( s \)-to-\( y \) edge labeled \( r \);
- there is a subject \( x' = x \) or \( x' \) initially spans to \( x \);
- there is a subject \( s' = s \) or \( s' \) terminally spans to \( s \); and
- there are islands \( I_1, \ldots, I_k \) connected by bridges, \( x' \) is in \( I_1 \), and \( s' \) is in \( I_k \)
Outline of Proof

1. $s$ has $r$ rights over $y$
2. $s'$ acquires $r$ rights over $y$ from $s$
   - Definition of terminal span
3. $x'$ acquires $r$ rights over $y$ from $s'$
   - Repeated application of sharing among vertices in islands, passing rights along bridges
4. $x'$ gives $r$ rights over $y$ to $x$
   - Definition of initial span
Interpretation

- Access control matrix is generic
  - Can be applied in any situation
- Take-Grant has specific rules, rights
  - Can be applied in situations matching rules, rights
- What states can evolve from a system that is modeled using the Take-Grant Protection Model?
Take-Grant Generated Systems

Theorem: Let $G_0$ be a protection graph with 1 subject and no edges. Let $R$ be a set of rights. Then $G_0 \vdash^* G$ if, and only if,

- $G$ is a finite, directed graph consisting of subjects, objects, and edges;
- the edges are labeled from a non-empty subset of $R$; and
- at least 1 vertex in $G$ has no incoming edges
Proof (1)

⇒: By construction; let $G$ be the final graph in the theorem

- Let $x_1, \ldots, x_n$ be subjects in $G$
- Let $x_1$ have no incoming edges
- Let $\alpha = R$

Construct $G'$ as follows:

1. Do “$x_1$ creates ($\alpha \cup \{g\}$ to) new subject $x_i$”
2. For all $(x_i, x_j)$ where $x_i$ has a right over $x_j$, do “$x_1$ grants ($\alpha$ to $x_j$) to $x_i$”
3. Let $\beta$ be the rights $x_i$ has over $x_j$ in $G$; then do “$x_1$ removes ($($($\alpha \cup \{g\}$) - $\beta$) to $x_j$)"

Now $G'$ is the desired $G$
Proof (2)

\( \iff \) Let \( v \) be the initial subject, and \( G_0 \vdash^* G \)

- Inspection of rules gives:
  - \( G \) is finite;
  - \( G \) is a directed graph;
  - Subjects and objects only; and
  - All edges are labeled with nonempty subsets of \( R \)

- Limits of rules:
  - None allows vertices to be deleted, so \( v \) is in \( G \)
  - None adds \textit{incoming} edges to vertices without any incoming edges, so \( v \) has no incoming edges.
Example: Shared Buffer

Goal: \( p, q \) to communicate through shared buffer \( b \) controlled by trusted entity \( s \)
Example: Shared Buffer

Goal: \( p, q \) to communicate through shared buffer \( b \) controlled by trusted entity \( s \)

1. \( s \) creates (\( \{r, w\} \) to) new object \( b \)
Example: Shared Buffer

Goal: \( p, q \) to communicate through shared buffer \( b \) controlled by trusted entity \( s \)

1. \( s \) creates \((\{r, w\} \text{ to})\) new object \( b \)
2. \( s \) grants \((\{r, w\} \text{ to } b)\) to \( p \)
Example: Shared Buffer

Goal: \( p, q \) to communicate through shared buffer \( b \) controlled by trusted entity \( s \)

1. \( s \) creates \( \{r, w\} \) to new object \( b \)
2. \( s \) grants \( \{r, w\} \) to \( b \) to \( p \)
3. \( s \) grants \( \{r, w\} \) to \( b \) to \( q \)
Stealing rights

**can-steal** Predicate

can-steal\((r, x, y, G_0)\) holds if, and only if, there is no edge from \(x\) to \(y\) labeled \(r\) in \(G_0\), and the following hold simultaneously:

- there is an edge from \(x\) to \(y\) labeled \(r\) in \(G\);
- there is a sequence of rule applications \(\rho_1, \ldots, \rho_n\) such that \(G_{i-1} \vdash_{\rho_i} G_i\); and
- for all vertices \(v, w\) in \(G_{i-1}\), if there is an edge from \(v\) to \(y\) in \(G_0\) labeled \(r\), then \(\rho_i\) is *not* of the form “\(v\) grants \((r\ to \(y\))\ to \(w\)”
Example of Stealing

\[ \text{can-steal}(\alpha, s, w, G_0) \]
Stealing rights

Example of Stealing

\[
\text{can-steal}(\alpha, s, w, G_0):
\]

1. u grants (t to v) to s
Example of Stealing

\[ can\text{-}steal(\alpha, s, w, G_0) : \]
1. \( u \) grants \((t\text{ to } v)\) to \( s \)
2. \( s \) takes \((t\text{ to } x)\) from \( v \)

\[ \]
Example of Stealing

\[ \text{can\textcdotsteal}(\alpha, s, w, G_0): \]

1. **u** grants \((t \text{ to } v)\) to **s**
2. **s** takes \((t \text{ to } x)\) from **v**
3. **s** takes \((t \text{ to } u)\) from **x**
Example of Stealing

\[
can\text{-steal}(\alpha, s, w, G_0):
\]

1. \(u\) grants \((t \text{ to } v)\) to \(s\)
2. \(s\) takes \((t \text{ to } x)\) from \(v\)
3. \(s\) takes \((t \text{ to } u)\) from \(x\)
4. \(s\) takes \((\alpha \text{ to } w)\) from \(u\)
can·steal Theorem

can·steal(\(\alpha, \ x, \ y, \ G_0\)) holds if, and only if, the following hold simultaneously:

- there is no edge from \(x\)-to-\(y\) labeled \(\alpha\) in \(G_0\);
- there is a subject \(x' = x\) or \(x'\) initially spans to \(x\);
- there is a vertex \(s\) with an edge to \(y\) labeled \(\alpha\) in \(G_0\); and
- \(can\cdotshare(t, \ x', \ s, \ G_0)\) holds
Proof (1)

⇒: Assume all four conditions hold

- If \( x \) a subject:
  - \( x \) gets \( t \) rights to \( s \) (last condition); then takes \( \alpha \) to \( y \) from \( s \) (third condition)

- If \( x \) an object:
  - \( \text{can-share}(t, x', s, G_0) \) holds
  - If \( x' \) has no \( \alpha \) edge to \( y \) in \( G_0 \), \( x' \) takes \( (\alpha \) to \( y) \) from \( s \) and grants it to \( x \)
  - If \( x' \) has an edge to \( y \) in \( G_0 \), \( x' \) creates surrogate \( x'' \), gives it \( (t \) to \( s) \) and \( (g \) to \( x'') \); then \( x'' \) takes \( (\alpha \) to \( y) \) and grants it to \( x \)
Proof (2)

\[ \iff: \text{Assume } \text{can}\cdot\text{steal}(\alpha, x, y, G_0) \text{ holds} \]

- First two conditions are immediate from definition of \text{can}\cdot\text{share}, \text{can}\cdot\text{steal}
- Third condition is immediate from theorem of conditions for \text{can}\cdot\text{share}
- Fourth condition: let \( \rho \) be a minimal length sequence of rule applications deriving \( G_n \) from \( G_0 \)
  - Let \( i \) be the smallest index such that \( G_{i-1} \vdash_{\rho_i} G_i \) that adds \( \alpha \) from some \( p \) to \( y \) in \( G_i \)
  - What rule is \( \rho_i \)?
Proof (3)

- Not remove or create rule
  - y exists already
- Not grant rule
  - $G_i$ is the first graph in which an edge labeled $\alpha$ to $y$ is added, so by definition of $\text{can\cdot share}$, it cannot be a grant
- Therefore $\rho_i$ must be a take rule, so $\text{can\cdot share}(t, p, s, G_0)$ holds
  - By earlier theorem, there is a subject $s'$ such that $s' = s$ or $s'$ terminally spans to $s$
  - Also, sequence of islands $l_1, \ldots, l_n$ with $x' \in l_1$, $s' \in l_n$
- Now consider what $s$ is
Proof (4)

- If $s$ object, $s' \neq s$
  - If $s'$, $p$ in same island, take $p = s'$; the $can\cdot share(t, x, s, G_0)$ holds
  - If they are not, the sequence is minimal, contradicting assumption
  - So choose $s'$ in same island as $p$
Stealing rights

Proof (5)

If \( s \) subject, \( p \in I_n \)

- If \( p \notin G_0 \), there is a subject \( q \) such that \( \text{can-share}(t, q, s, G_0) \) holds
  - \( s \in G_0 \) and none of the rules add new labels to incoming edges on existing vertices

- As \( s \) owns \( \alpha \) rights to \( y \) in \( G_0 \), two cases arise:
  - If \( s = q \), replace “\( s \) grants (\( \alpha \) to \( y \)) to \( q \)” with the sequence:
    - \( p \) takes (\( \alpha \) to \( y \)) from \( s \)
    - \( p \) takes (\( g \) to \( q \)) from \( s \)
    - \( p \) grants (\( \alpha \) to \( y \)) to \( q \)
  - If \( s = q \), you only need the first
Conspiracy

- Minimize number of actors to generate a witness for $can\cdot share(\alpha, \ x, \ y, \ G_0)$
  - Actor is defined as $x$ such that $x$ initiates $\rho_i$
- Access set describes the “reach” of a subject
- Deletion set is set of vertices that cannot be involved in a transfer of rights
- Build conspiracy graph to capture how rights flow, and derive actors from it
Access Set

- **Access set** $A(x)$ **with focus** $x$: set of vertices
  - $\{x\}$
  - $\{y \mid x \text{ initially spans to } y\}$
  - $\{y \mid x \text{ terminally spans to } y\}$

- Idea is that vertex at focus can give rights to, or acquire rights from, a vertex in access set