ECS 235B, Lecture 2

January 9, 2019

Access Control Matrix

Attributes

- *attribute*: variable of a specific data type associated with an entity
- *att*(*o*): set of attribute values associated with *o*, called the *attribute value tuple* of *o*
  - Each attribute is written $o.a_i$, with value v drawn from set Va_i
- *attribute predicate*: boolean expression built from attributes and constants with appropriate operation and relation symbols
  - Unary predicate: built from one attribute
  - Binary predicate: built from two attributes
  - Can have as many attributes in a predicate as needed
  - Example: *Alice.credit* ≥ $100.00

Attribute Based Access Control Matrix (ABAM)

- Change access control matric so rows correspond to subject and its attributes, and object and its attributes
- Note access control matrix discussed previously is special case
  - Just make the attribute sets be empty

Primitive Operations

- **enter**,**delete** as before
- **create subject** *s* **with attribute tuple** *att*(*s*): create subject *s* with given attribute tuple; additionally, add an identity attribute with a unique value
- **create object** *o* **with attribute tuple** *att*(*o*): create object *o* with given attribute tuple; additionally, add an identity attribute with a unique value
- **destroy** as before except it also deletes. the associated attribute tuple
- **update attribute** $o.a_i$: update $att(o) = (v_1, \ldots, v_i, \ldots, v_n)$ to $att(o)' = (v_1, \ldots, v_i', \ldots, v_n)$, where $v_i, v_i' \in Va_i$, and $v_i \neq v_i'$

Commands

- Like previous commands, except that conditions may include attribute predicates
- Let *p* give *q* *r* rights over *f*, if *p* owns *f* and value of *p*'s attribute *jobcode* is between 3 and 5 inclusive

```
command grant•read•file•attribute•3to5(p, f, q)
  if own in A[p, f] and 3 ≤ p.jobcode and p.jobcode ≤ 5
  then
    enter r into A[q, f];
end
```

Foundational Results

Overview

- Safety Question
- HRU Model
- Take-Grant Protection Model
- SPM, ESPM
  - Multiparent joint creation
- Expressive power
- Typed Access Matrix Model
- Comparing properties of models

What Is “Secure”?

- Adding a generic right *r* where there was not one is “leaking”
  - In what follows, a right leaks if it was not present *initially*
  - Alternately: not present *in the previous state* (not discussed here)
- If a system *S*, beginning in initial state s_0, cannot leak right *r*, it is *safe with respect to the right r*
  - Otherwise it is called *unsafe with respect to the right r*

Safety Question

- Is there 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 **create**s 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

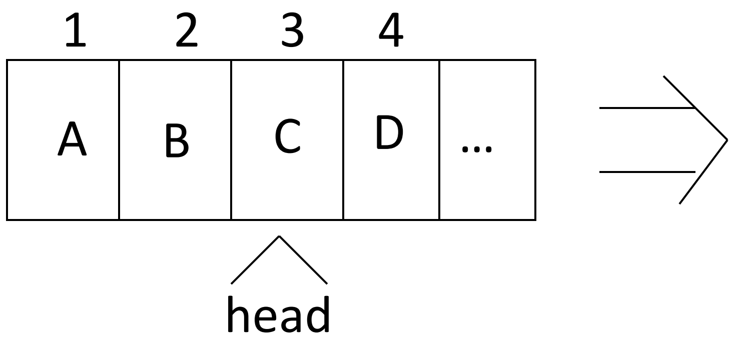
- Answer: *no*
- Sketch of proof:

  Reduce halting problem to safety problem

  Turing Machine review:

  - Infinite tape in one direction
  - States K, symbols M; distinguished blank b
  - Transition function $\delta(k, m) = (k', m', \text{L})$ 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

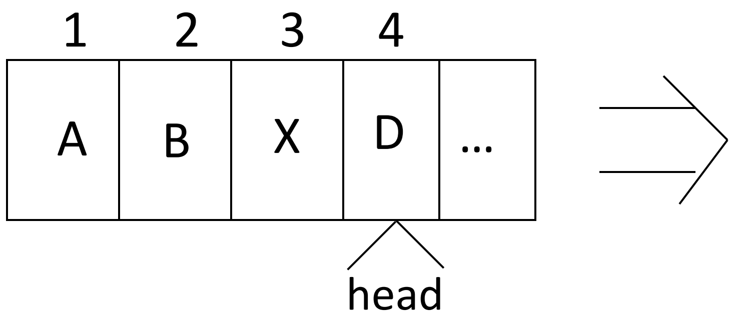
Mapping



Current state is *k*

	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C *k*	*own*	
s_4				D *end*	

Mapping



	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 *end*	

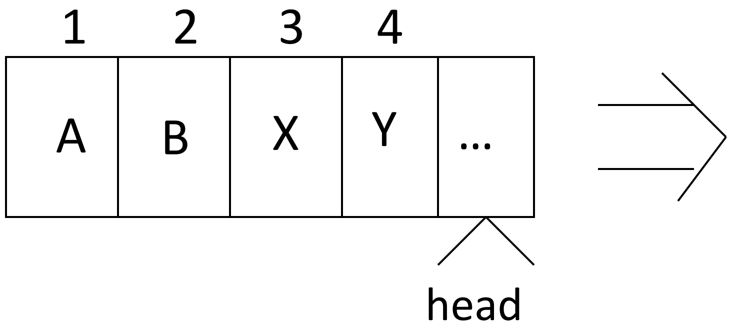
After $\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ where k is the current state and k_1 the next state

Command Mapping

- $\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ at intermediate becomes

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

Mapping



	s_1	s_2	s_3	s_4	s_5
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				Y	*own*
s_5					b k_2 end

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

Command Mapping

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

```
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];
 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];
 create subject s_5;
 enter own into A[s_4,s_5];
 enter end into A[s_5,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; then safety question is undecidable
    - Systems are monotonic
- Safety question for biconditional protection systems is decidable
- 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)

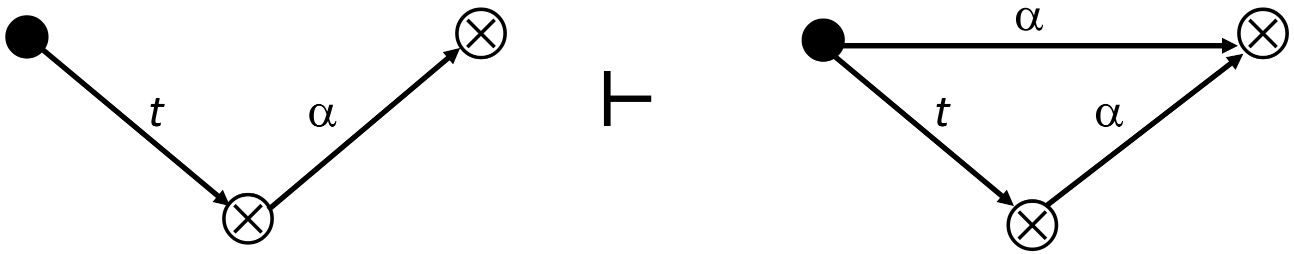
$G \vdash_x G'$ apply a rewriting rule *x* (witness) to *G* to get *G′*

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

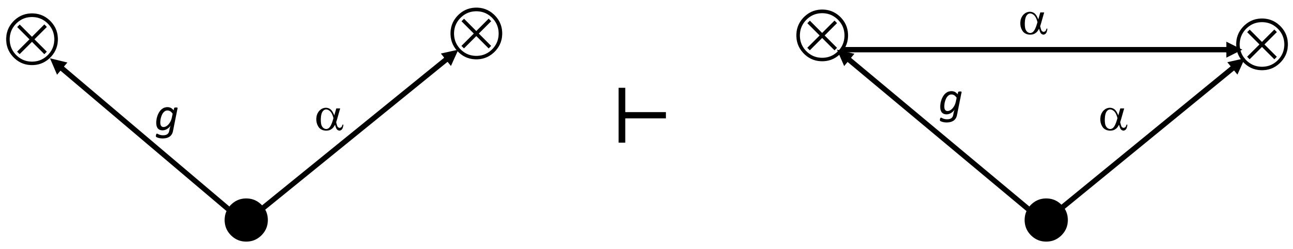
$R = \{ t, g, r, w, \ldots \}$ set of rights

Rules

take



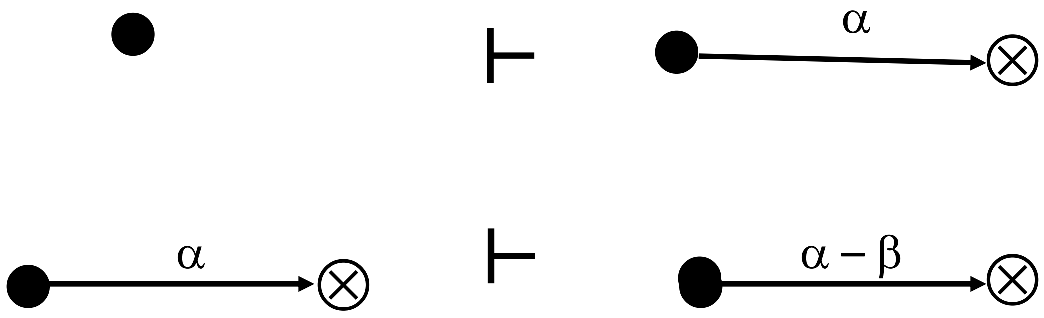
grant



More Rules

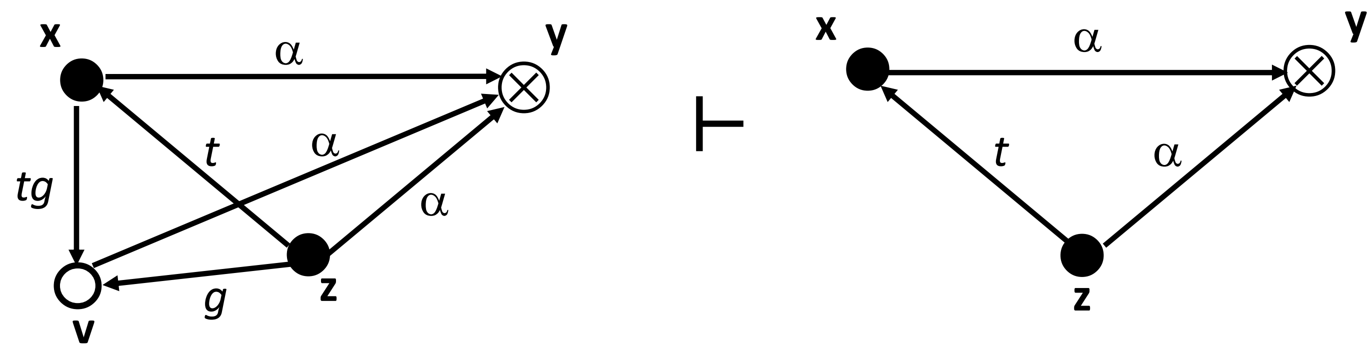
create

remove



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 (α to ***y***) to ***v***
4. ***x*** takes (α 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 $\{ \vec{t}^* \vec{g} \} \cup \{ \nu \}$
  - 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 $\{ \vec{t}^* \} \cup \{ \nu \}$
  - Means **x** can acquire any rights **y** has

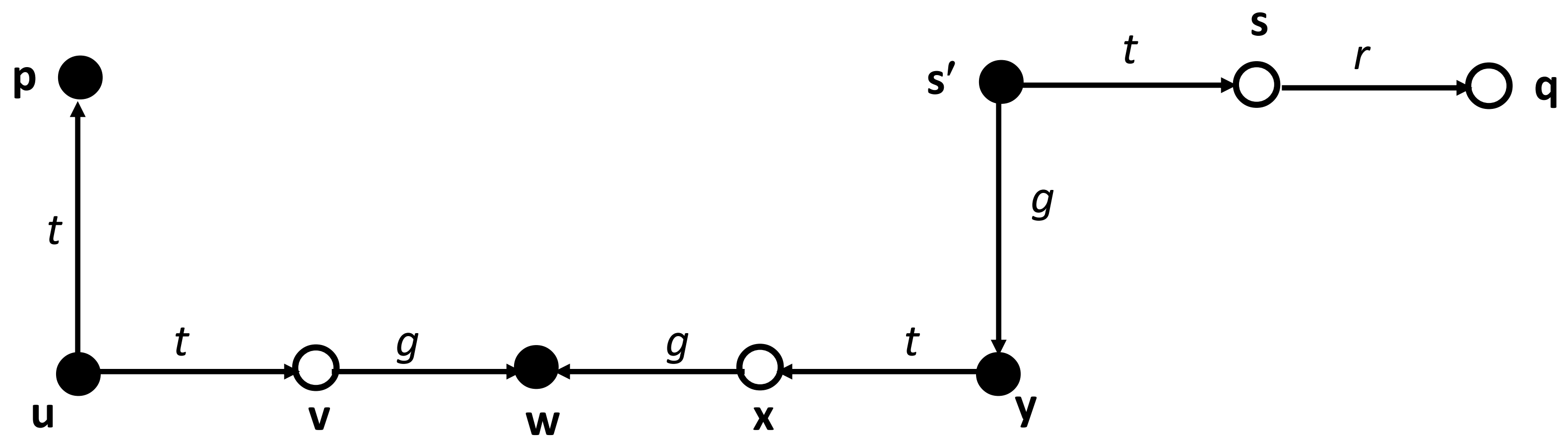
Bridges

- bridge: *tg*-path between subjects **x**, **y**, with associated word in

$$\{ \overrightarrow{t}^*, \overrightarrow{t}^*, \overrightarrow{t}^* \overleftarrow{g}\, \overleftarrow{t}^*, \overrightarrow{t}^* \overrightarrow{g}\, \overleftarrow{t}^* \}$$

  - rights can be transferred between the two endpoints
  - *not* an island as intermediate vertices are objects

Example



- islands { **p**, **u** } { **w** } { **y**, **s′** }
- bridges **uvw**; wxy
- initial span **p** (associated word ν)
- terminal span **s′s** (associated word $\vec{t}$)

can•share Predicate

Definition:

- *can•share*(r, **x**, **y**, G_0) if, and only if, there is a sequence of protection graphs G_0, …, G_n such that $G_0 \vdash^* G_n$ using only *de jure* rules and in G_n there is an edge from **x** to **y** labeled r.

can•share Theorem

- *can•share*(r, **x**, **y**, G_0) 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 initially spans to **x**
  - There is a subject **s′** = **s** or terminally spans to **s**
  - There are islands $I_1, \dots, I_k$ connected by bridges, and **x′** in I_1 and **s′** in I_k

Outline of Proof

- **s** has *r* rights over **y**
- **s**′ acquires *r* rights over **y** from **s**
  - Definition of terminal span
- **x**′ acquires *r* rights over **y** from **s**′
  - Repeated application of sharing among vertices in islands, passing rights along bridges
- **x**′ gives *r* rights over **y** to **x**
  - Definition of initial span

Example Interpretation

- ACM is generic
  - Can be applied in any situation
- Take-Grant has specific rules, rights
  - Can be applied in situations matching rules, rights
- Question: what states can evolve from a system that is modeled using the Take-Grant Model?

Take-Grant Generated Systems

- Theorem: G_0 protection graph with 1 vertex, no edges; R set of rights. Then $G_0 \vdash^* G$ iff:
  - *G* finite directed graph consisting of subjects, objects, edges
  - Edges labeled from nonempty subsets of *R*
  - At least one vertex in *G* has no incoming edges

Outline of Proof

$\Rightarrow$: By construction; G final graph in theorem

- Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be subjects in G
- Let $\mathbf{x}_1$ have no incoming edges

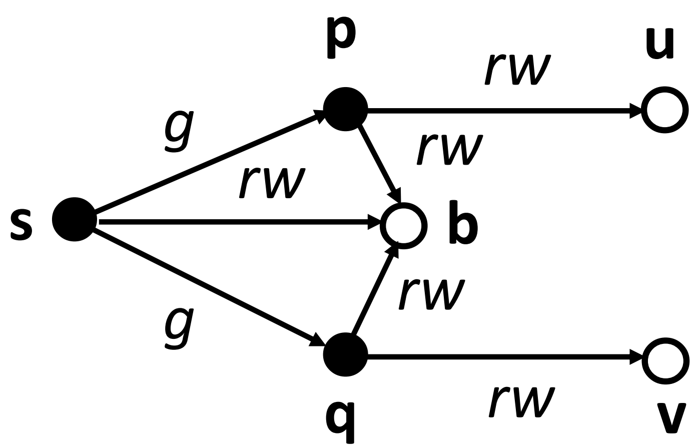
- Now construct G' as follows:
  1. Do "$\mathbf{x}_1$ creates ($\alpha \cup \{ g \}$ to) new subject $\mathbf{x}_i$"
  2. For all ($\mathbf{x}_i$, $\mathbf{x}_j$) where $\mathbf{x}_i$ has a rights over $\mathbf{x}_j$, do "$\mathbf{x}_1$ grants (α to $\mathbf{x}_j$) to $\mathbf{x}_i$"
  3. Let β be rights $\mathbf{x}_i$ has over $\mathbf{x}_j$ in G. Do "$\mathbf{x}_1$ removes (($\alpha \cup \{ g \} - \beta$ to) $\mathbf{x}_j$"
- Now G' is desired G

Outline of Proof

$\Leftarrow$: Let **v** be initial subject, and $G_0 \vdash^* G$

- Inspection of rules gives:
  - *G* is finite
  - *G* is a directed graph
  - Subjects and objects only
  - All edges labeled with nonempty subsets of *R*
- Limits of rules:
  - None allow vertices to be deleted so **v** in *G*
  - None add incoming edges to vertices without 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**
  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**