Lecture for January 11, 2016

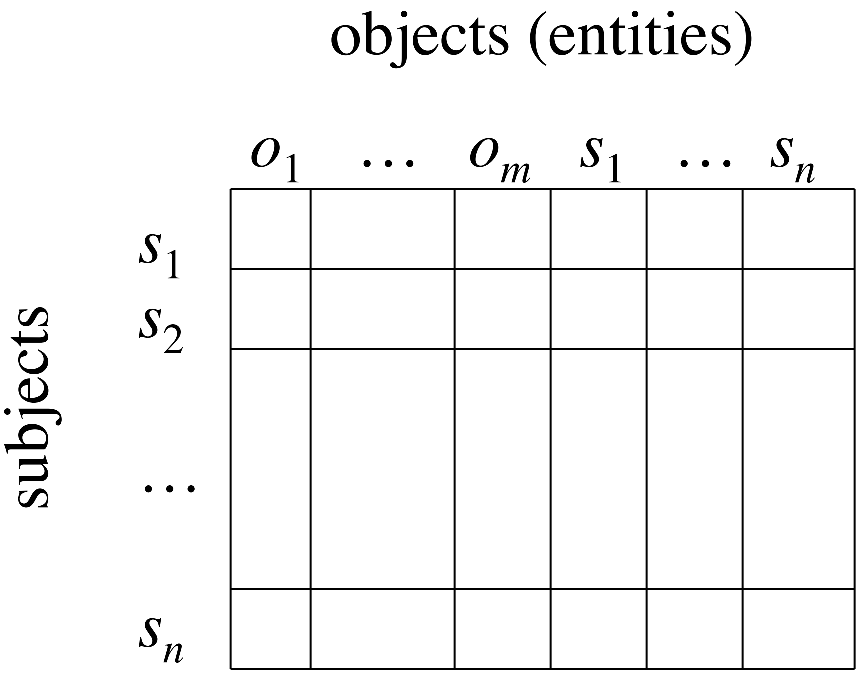
ECS 235A

UC Davis

Overview

- Protection state of system
  – Describes current settings, values of system relevant to protection
- Access control matrix
  – Describes protection state precisely
  – Matrix describing rights of subjects
  – State transitions change elements of matrix

Description



- Subjects $S = \{ s_1,\ldots, s_n \}$
- Objects $O = \{ o_1,\ldots, o_m \}$
- Rights $R = \{ r_1,\ldots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

Example 1

- Processes p, q
- Files f, g
- Rights r, w, x, a, o

	f	g	p	q
p	rwo	r	$rwxo$	w
q	a	ro	r	$rwxo$

Example 2

- Host names *telegraph*, *nob*, *toadflax*
- Rights *own*, *ftp*, *nfs*, *mail*

	telegraph	*nob*	*toadflax*
telegraph	*own*	*ftp*	*ftp*
nob		*ftp*, *mail*, *nfs*, *own*	*ftp*, *nfs*, *mail*
toadflax		*ftp*, *mail*	*ftp*, *mail*, *nfs*, *own*

State Transitions

- Change the protection state of system
- $\vdash$ represents transition
  - $X_i \vdash_\tau X_{i+1}$: command τ moves system from state X_i to X_{i+1}
  - $X_i \vdash^* Y$: a sequence of commands moves system from state X_i to Y
- Commands often called *transformation procedures*

Primitive Operations

- **create subject** *s*; **create object** *o*
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** *s*; **destroy object** *o*
  - Deletes row, column from ACM; deletes column from ACM
- **enter** *r* **into** $A[s, o]$
  - Adds *r* rights for subject *s* over object *o*
- **delete** *r* **from** $A[s, o]$
  - Removes *r* rights from subject *s* over object *o*

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O')\ [a'[s, y] = \varnothing]$, $(\forall x \in S')\ [a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)\ [a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o
- Postconditions:
  - $S' = S, O' = O \cup \{ o \}$
  - $(\forall x \in S') [a'[x, o] = \emptyset]$
  - $(\forall x \in S)(\forall y \in O) [a'[x, y] = a[x, y]]$

Add Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: enter r into $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Delete Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** r **from** $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** s
- Postconditions:
  - $S' = S - \{ s \}, O' = O - \{ s \}$
  - $(\forall y \in O') \, [a'[s, y] = \varnothing], (\forall x \in S') \, [a'[x, s] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O') \, [a'[x, y] = a[x, y]]$

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** o
- Postconditions:
  - $S' = S, O' = O - \{ o \}$
  - $(\forall x \in S') [a'[x, o] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O') [a'[x, y] = a[x, y]]$

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

Conditional Commands

- Let *p* give *q* *r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
            enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

Multiple Conditions

- Let *p* give *q* *r* and *w* rights over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant•read•file•2(p, f, q)
    if own in A[p, f] and c in A[p, q]
    then
          enter r into A[q, f];
          enter w into A[q, f];
end
```

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.

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*
- 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

- 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:

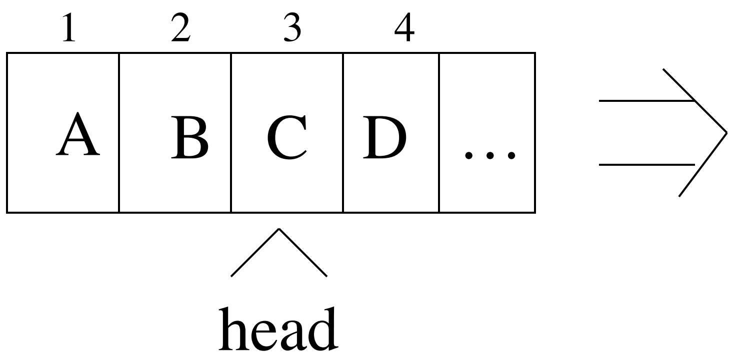
  – Infinite tape in one direction

  – States K, symbols M; distinguished blank b

  – Transition function $\delta(k, m) = (k', m', \mathrm{L})$ 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

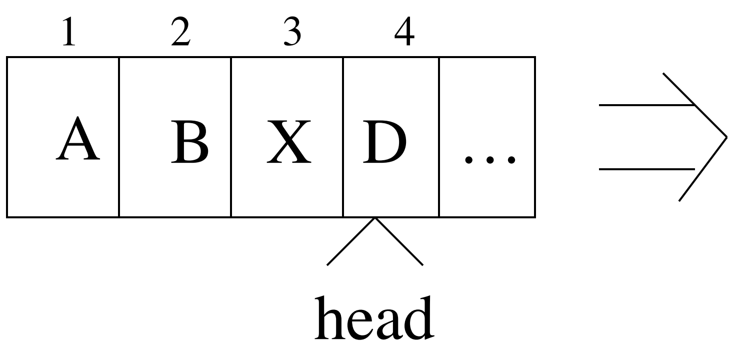
Mapping



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	

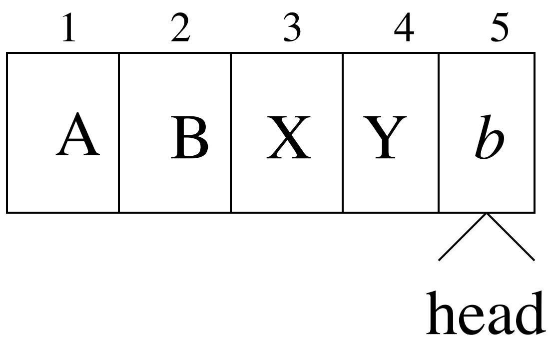
After $\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ where k is the current state and k_1 the next state

Command Mapping

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

command $\texttt{c}_{k,\texttt{C}}(s_3, s_4)$
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 $\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$ where k_1 is the current state and k_2 the next state

Command Mapping

$\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$ 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.

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, τ, A)
  - $\tau: O \rightarrow T$ specifies type of each object
  - If **X** subject, $\tau(\mathbf{X})$ in TS
  - If **X** object, $\tau(\mathbf{X})$ in $T - TS$

Create Rules

- Subject creation
  - **create subject** *s* **of type** *ts*
  - *s* must not exist as subject or object when operation executed
  - $ts \in TS$
- Object creation
  - **create object** *o* **of type** *to*
  - *o* must not exist as subject or object when operation executed
  - $to \in T - TS$

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s **of type** t
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(s) = t$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

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

Definitions

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

Cyclic Creates

command *cry•havoc*$(s_1 : u, s_2 : u, o_1 : v, o_2 : v, o_3 : w, o_4 : w)$

  create subject s_1 **of type** u;

  create object o_1 **of type** v;

  create object o_3 **of type** w;

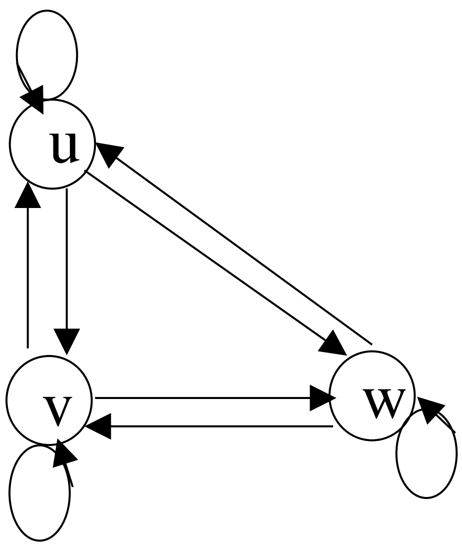
  enter r **into** $a[s_2, s_1]$;

  enter r **into** $a[s_2, o_2]$;

  enter r **into** $a[s_2, o_4]$

end

Creation Graph



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

Acyclic Creates

command *cry•havoc*$(s_1 : u, s_2 : u, o_1 : v, o_3 : w)$

create object o_1 **of type** v;

create object o_3 **of type** w;

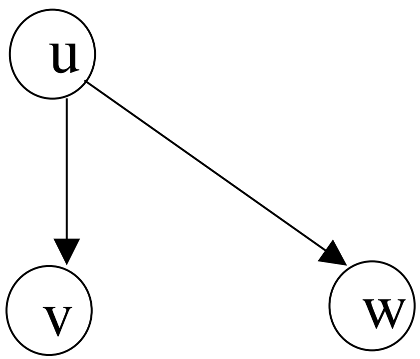
enter r **into** $a[s_2, s_1]$;

enter r **into** $a[s_2, o_1]$;

enter r **into** $a[s_2, o_3]$

end

Creation Graph



- *v*, *w* child types
- *u* parent type
- Graph: lines from parent types to child types
- This one has no cycles

Theorems

- Safety decidable for systems with acyclic MTAM schemes
  - In fact, it's *NP-hard*
- Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  - "Ternary" means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM