# ECS 235B, Lecture 1

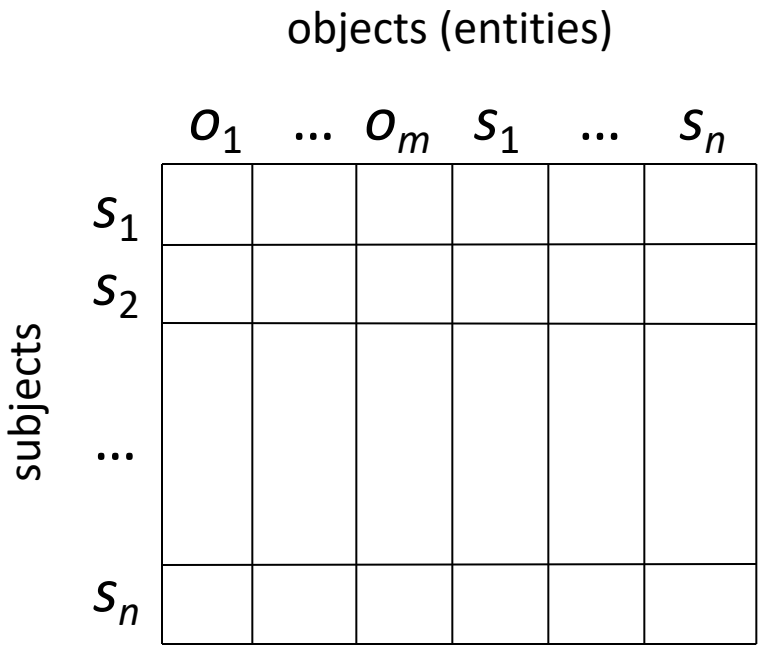
January 7, 2019

# Access Control Matrix

# Overview

- Access Control Matrix Model
  - Boolean Expression Evaluation
  - History
- Protection State Transitions
  - Commands
  - Conditional Commands
- Special Rights
  - Principle of Attenuation of Privilege

# 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* |

# Example 3

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights *+*, *–*, *call*

| | *counter* | *inc_ctr* | *dec_ctr* | *manage* |
|---|---|---|---|---|
| *inc_ctr* | *+* | | | |
| *dec_ctr* | *–* | | | |
| *manager* | | *call* | *call* | *call* |

# Boolean Expression Evaluation

- ACM controls access to database fields
  - Subjects have attributes
  - Verbs define type of access
  - Rules associated with objects, verb pair
- Subject attempts to access object
  - Rule for object, verb evaluated, grants or denies access

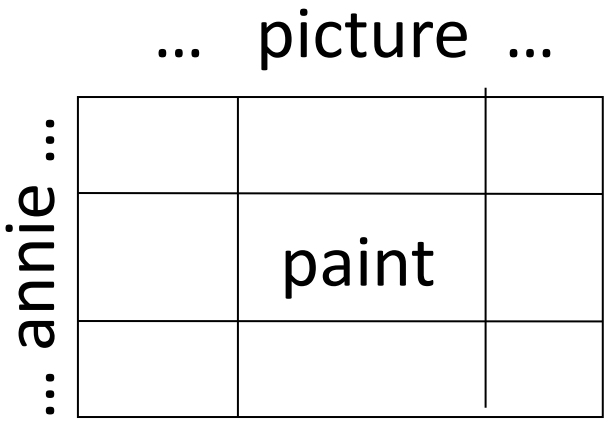
# Example

- Subject annie
  - Attributes *role* (artist), *group* (creative)
- Verb paint
  - Default 0 (deny unless explicitly granted)
- Object picture
  - Rule:

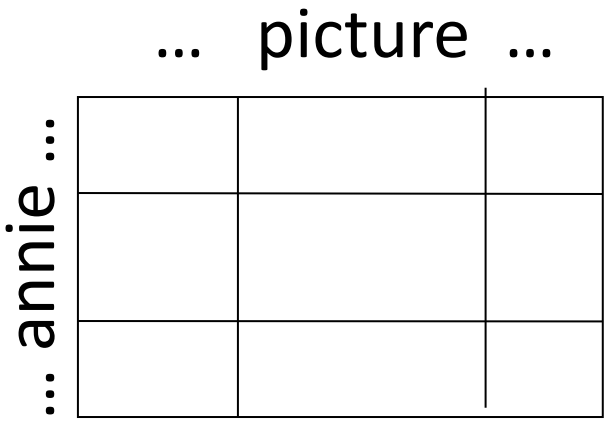
    paint: ‘artist’ in subject.role and
    ‘creative’ in subject.groups and
    time.hour ≥ 0 and time.hour ≤ 4

# ACM at 3AM and 10AM

At 3AM, time condition met
ACM is:




At 10AM, time condition not met
ACM is:



# History

- Problem: what a process has accessed may affect what it can access now
- Example: procedure in a web applet can access other procedures depending on what procedures it has already accessed
  - *S* set of *static rights* associated with procedure
  - *C* set of current rights associated with each executing process
  - When process calls procedure, rights are $S \cap C$

# Example Program

```
// This routine has no filesystem access rights
// beyond those in a limited, temporary area
procedure helper_proc()
        return sys_kernel_file

// But this has the right to delete files
program main()
        sys_load_file(helper_proc)
        tmp_file = helper_proc()
        sys_delete_file(tmp_file)
```

- *sys_kernel_file* contains system kernel
- *tmp_file* is in limited area that *helper_proc*() can access

# Before *helper_proc* Called

- Static rights of program

| | *sys_kernel_file* | *tmp_file* |
|---|---|---|
| *main* | delete | delete |
| *helper_proc* | | delete |

- When program starts, current rights:

| | *sys_kernel_file* | *tmp_file* |
|---|---|---|
| *main* | delete | delete |
| *helper_proc* | | delete |
| *process* | delete | delete |

# After *helper_proc* Called

- Process rights are intersection of static, previous "current" rights:

| | *sys_kernel_file* | *tmp_file* |
|---|---|---|
| *main* | delete | delete |
| *helper_proc* | | delete |
| *process* | | delete |

# State Transitions

- Change the protection state of system
- $|-$ represents transition
  - $X_i \; |-_{\tau} \; X_{i+1}$: command $\tau$ moves system from state $X_i$ to $X_{i+1}$
  - $X_i \; |-^{*} \; Y$: 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:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall x \in S')\ [A'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)\ [A'[x, y] = A[x, y]]$

# 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 Flag and Right

- Allows possessor to give rights to another
- Often attached to a right (called a *flag*), 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 increase your rights, or 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.

# Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions