Chapter 13: Design Principles

- Overview
- Principles
  - Least Privilege
  - Fail-Safe Defaults
  - Economy of Mechanism
  - Complete Mediation
  - Open Design
  - Separation of Privilege
  - Least Common Mechanism
  - Psychological Acceptability

Overview

- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access
  - Inhibit communication

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
  - Minimal protection domain

Fail-Safe Defaults

- Default action is to deny access
- If action fails, system as secure as when action began

Economy of Mechanism

- Keep it as simple as possible
  - KISS Principle
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

Complete Mediation

- Check every access
- Usually done once, on first action
  - UNIX: Access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access

Open Design

- Security should not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - "Security through obscurity"
  - Does not apply to information such as passwords or cryptographic keys

Separation of Privilege

- Require multiple conditions to grant privilege
  - Separation of duty
  - Defense in depth

Least Common Mechanism

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Psychological Acceptability

- Security mechanisms should not add to difficulty of accessing resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here

Key Points

- Principles of secure design underlie all security-related mechanisms
- Require:
  - Good understanding of goal of mechanism and environment in which it is to be used
  - Careful analysis and design
  - Careful implementation

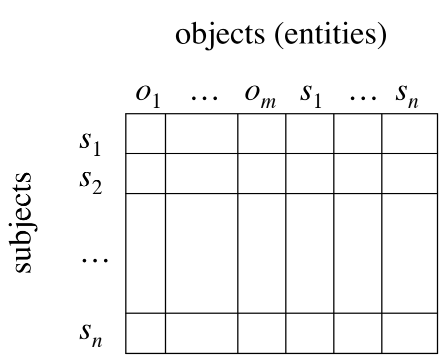
Chapter 2: Access Control Matrix

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

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

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

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*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), groups (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 < 5

ACM at 3AM and 10AM

At 3AM, time condition met; ACM is:

… picture …

… annie …		paint	

At 10AM, time condition not met; ACM is:

… picture …

… annie …			

History

Database:

name	**position**	**age**	**salary**
Alice	teacher	45	$40,000
Bob	aide	20	$20,000
Cathy	principal	37	$60,000
Dilbert	teacher	50	$50,000
Eve	teacher	33	$50,000

Queries:

1. sum(salary, “position = teacher”) = 140,000
2. sum(salary, “age > 40 & position = teacher”) should not be answered (deduce Eve’s salary)

ACM of Database Queries

O_i = { objects referenced in query i }

$f(o_i)$ = { read } for $o_j \in O_i$, if $|\cap_{j=1,\ldots,i} O_j| < 2$

$f(o_i) = \varnothing$ for $o_j \in O_i$, otherwise

1. O_1 = { Alice, Dilbert, Eve } and no previous query set, so:

   A[asker, Alice] = f(Alice) = { read }

   A[asker, Dilbert] = f(Dilbert) = { read }

   A[asker, Eve] = f(Eve) = { read }

   and query can be answered

April 6, 2004 ECS 235 Slide #21

But Query 2

From last slide:

$f(o_i)$ = { read } for $o_j \in O_i$, if $|\cap_{j=1,\ldots,i} O_j| < 2$

$f(o_i) = \varnothing$ for $o_j \in O_i$, otherwise

2. O_2 = { Alice, Dilbert } but $| O_2 \cap O_1 | = 2$ so

   A[asker, Alice] = f(Alice) = $\varnothing$

   A[asker, Dilbert] = f(Dilbert) = $\varnothing$

   and query cannot be answered

April 6, 2004 ECS 235 Slide #22

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}
  - $X_i \vdash^* X_{i+1}$: a sequence of commands moves system from state X_i to X_{i+1}
- 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 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.

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

Chapter 3: Foundational Results

- Overview
- Harrison-Ruzzo-Ullman result
  - Corollaries
- Take-Grant Protection Model
- SPM and successors

Overview

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

What Is "Secure"?

- Adding a generic right *r* where there was not one is "leaking"
- 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

- 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*?
  - 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 $k \leq n(s+1)(o+1)$