CS590U
Access Control: Theory and Practice

Lecture 9 (February 7)

Formalizing Access Matrices: Graham-Denning and Harrison-Ruzzo-Ullman

History of Access Matrices

- Lampson’1971
  - “Protection”
- Refined by Graham and Denning’1972
  - “Protection---Principles and Practice”
- Harrison, Ruzzo, and Ullman’1976
  - “Protection in Operating Systems”

Access Matrix

- A set of subjects S
- A set of objects O
- A set of rights R
- An access control matrix
  - one row for each subject
  - one column for each subject/object
  - elements are right of subject on another subject or object

The Graham-Denning Work

- Based on access matrices
- Focuses on access control within an operating system
- Explores various possibilities of discretionary access control

Seven Levels of Protection / Separation

1. No sharing at all
2. Sharing copies of programs or data files
3. Sharing originals of programs or data files
4. Sharing programming systems or subsystems
5. Permitting the cooperation of mutually suspicious subsystems, e.g., debugging or proprietary subsystems
6. Providing memory-less subsystems
7. Providing “certified” subsystems

Elements in Graham-Denning

- Objects: have unique identifier
- Subjects
  - a subject is a pair (process, domain)
  - forging a subject identifier is impossible (authentication)
- Protection state
  - modeled using an access matrix (can also be represented as a graph)
- No modeling of actual accesses (only access permissions)
  - whether this is sufficient depends on the properties to be studied

Special Rights in Graham-Denning Model

- Each subject/object has an owner
- Each subject has a controller (which may be itself)
- A right may be transferable or nontransferable

	Objects					
Subjects	S_1	S_2	S_3	O_1	O_2	O_3
S_1	control			owner	read write	
S_2		control	read*			execute
S_3			control		owner	

Eight Commands in Graham-Denning Model

1. subject x creates object o
   - no precondition
   - add column for o
   - place `owner' in A[x,o]

2. subject x creates subject s
   - no precondition
   - add row and column for s
   - place `control', `owner' in A[x,s]

Eight Commands in Graham-Denning Model

3. subject x destroys object o
   - precondition: `owner’ in A[x,o]
   - delete column o

4. subject x destroys subject s
   - precondition: `owner’ in A[x,s]
   - delete row and column for s

Eight Commands in Graham-Denning Model

5. subject x grants a right r/r* on object o to subject s
   - precondition: `owner’ in A[x,o]
   - stores r/r* in A[s,o]

6. subject x transfers a right r/r* on object o to subject s
   - precondition: r* in A[x,o]
   - stores r/r* in A[s,o]

Eight Commands in Graham-Denning Model

7. subject x deletes right r/r* on object o from subject s
   - precondition: `control' in A[x,s] or `owner' in A[x,o]
   - delete r/r* from A[s,o]

Eight Commands in Graham-Denning Model

8. subject x checks what rights subject s has on object o [w := read s,o]
   - precondition: `control' in A[x,s] OR `owner' in A[x,o]
   - copy A[s,o] to w
- This does not affect the protection state.
   - policy review functions
   - useful when analyzing external behaviors of the protection system, not clear why needed in this paper

Messy Details

- Some requirements place additional constraints on state-transitions
  - Each subject is owner or controlled by at most one other subject
    - cannot transfer/grant owner right
  - It is undesirable for a subject to be `owner' of itself, for then it can delete other subjects' access to itself
  - [The relation "owner" defines naturally a tree hierarchy on subjects.]
    - What does it take to maintain the hierarchy?

Other possible extensions

- Transfer-only copy flags
- Limited-use access attributes
  - needs to model access to use this feature
- Allow a subject to obtain a right that its subordinate has.
- The notion of "indirect" right
  - S_2 has indirect right over S means that S_2 can access anything that S is allowed to access, but S_2 cann't take right from S
  - differs from basic notion of an access matrix

M.A. Harrison, W.L. Ruzzo, and J.D. Ullman: Protection in Operating Systems.

Communications of the ACM, August 1976.

Objectives of the HRU Work

- Provide a model that is sufficiently powerful to encode several access control approaches, and precise enough so that security properties can be analyzed
- Introduce the “safety problem”
- Show that the safety problem
  - is decidable in certain cases
  - is undecidable in general
  - is undecidable in monotonic case

Protection Systems

- A protection system has
  - a finite set R of generic rights
  - a finite set C of commands
- A protection system is a state-transition system
- To model a system, specify the following constants:
  - set of all possible subjects
  - set of all possible objects
  - R

The State of A Protection System

- A set O of objects
- A set S of subjects that is a subset of O
- An access control matrix
  - one row for each subject
  - one column for each object
  - each cell contains a set of rights

Commands: Examples

```
command GRANT_read(x1,x2,y)
    if `own' in [x1,y]
    then enter `read' into [x2,y]
end

command CREATE_object(x,y)
    create object y
    enter `own' into [x,y]
end
```

Syntax of a Command

- A command has the form

  command $a(X_1, X_2, \ldots, X_k)$

  if

  r_1 in (X_{s1}, X_{o1}) and … and r_m in (X_{sm}, X_{om})

  then

  op_1 … op_n

  end

  - $X_1,\ldots,X_k$ are formal parameters

Six Primitive Operations

- enter r into (X_s, X_o)
  - Condition: $X_s \in S$ and $X_o \in O$
  - r may already exist in (X_s, X_o)
- delete r from (X_s, X_o)
  - Condition: $X_s \in S$ and $X_o \in O$
  - r does not need to exist in (X_s, X_o)

Six Primitive Operations

- create subject X_s
  - Condition: $X_s \notin O$
- create object X_o
  - Condition: $X_o \notin O$
- delete subject X_s
  - Condition: $X_s \in S$
- delete object X_o
  - Condition: $X_o \in O$ and $X_o \notin S$

How Does State Transition Work?

- Given a protection system (R, C), state z_1 can reach state z_2 iff there is an instance of a command in C so that all conditions are true at state z_1 and executing the primitive operations one by one results in state z_2
  - a command is executed as a whole (similar to a transaction), if one step fails, then nothing changes

Example

- Given the following command
  - command α (x, y, z)

    enter r1 into (x,x)

    destroy subject x

    enter r2 into (y,z)

    end
- One can never use α(s,s,o) to change a state

The Safety Problem

- What do we mean by “safe”?
  - Definition 1: “access to resources without the concurrence of the owner is impossible”
  - Definition 2: “the user should be able to tell whether what he is about to do (give away a right, presumably) can lead to the further leakage of that right to truly unauthorized subjects”

Defining the Safety Problem

- "Suppose a subject s plans to give subjects s′ generic right r to object o. The natural question is whether the current access matrix, with r entered into (s′,o), is such that generic right r could subsequently be entered somewhere new."

Defining the Safety Problem

- To avoid a trivial “unsafe” answer because s himself can confer generic right r, we should in most circumstances delete s itself from the matrix. It might also make sense to delete from the matrix any other “reliable” subjects who could grant r, but whom s “trusts” will not do so.

Defining the Safety Problem

- It is only by using the hypothetical safety test in this manner, with “reliable” subjects deleted, that the ability to test whether a right can be leaked has a useful meaning in terms of whether it is safe to grant a right to a subject.

Definition of the Safety Problem in [HRU]

- Given a protection system and generic right r, we say that the initial configuration Q_0 is unsafe for r (or leaks r) if there is a configuration Q and a command α such that
  - Q is reachable from Q_0
  - α leaks r from Q
- We say Q_0 is safe for r if Q_0 is not unsafe for r.

Definition of Right Leakage in [HRU]

- We say that a command $\alpha(x1,\ldots,xk)$ leaks generic right r from Q if α, when run on Q, can execute a primitive operation which enters r into a cell of the access matrix which did not previously contain r.

End of Lecture 9

- Next lecture (Thursday Feb 9)
  - cancelled for David Patterson's distinguished lecture
- The one after next (Tuesday Feb 14)
  - Safety in HRU
  - Read the HRU paper before the lecture