CSCE 465 Computer & Network Security

Instructor: Abner Mendoza

Homework 4

- Due on Friday – hard deadline 11:59pm
- IF you did not use TCR: Indicate WHY in your report! TA will validate your reason.

Security Theory I: Access Control Matrix & Foundational results

Outline

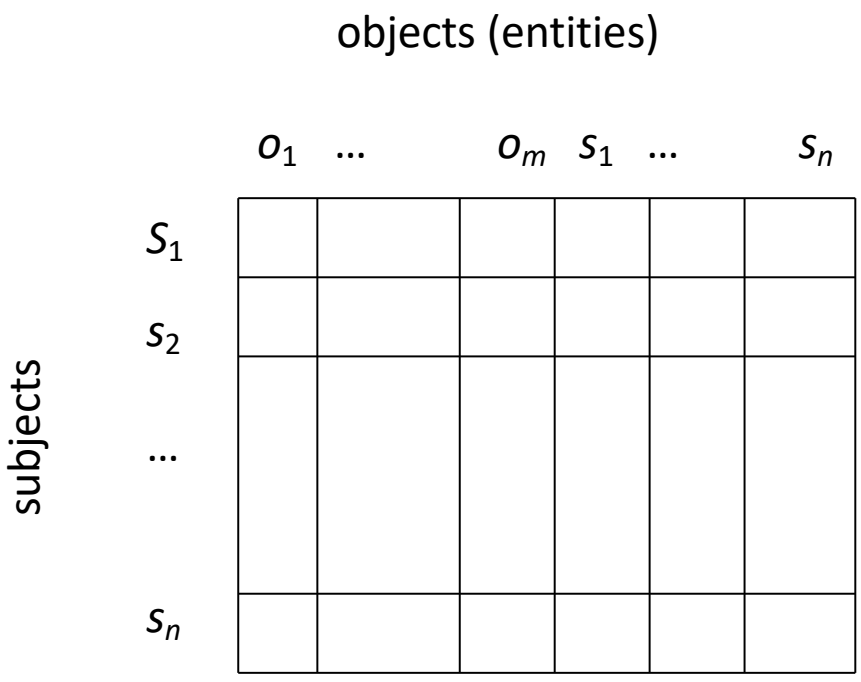
- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions
    - Commands
    - Conditional Commands

Access Control Matrix

Overview

- Protection state of system
  - Describes current settings and values of a system relevant to protection
- State of a system
  - A collection of the current values of all memory locations, storages, registers, etc.
  - A subset of this collection that deals with protection is the **protection state** of the system
- 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(ppend)*, *o(wn)*

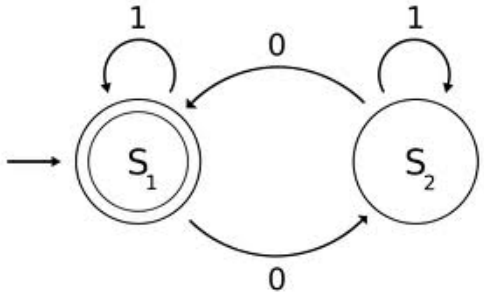
	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Example 2

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

	counter	*inc_ctr*	*dec_ctr*	*manager*
inc_ctr	*+*			
dec_ctr	*–*			
manager		*call*	*call*	*call*

State Transitions



- Change the protection state of system
- State transitions due to commands
- $|-$ represents transition
  - $X_i \; |-_{\tau} \; X_{i+1}$: command τ moves system from state X_i to X_{i+1}
  - $X_i \; |-^{*} \; X_{i+1}$: a sequence of commands moves system from state X_i to X_{i+1}
- Commands often called *transformation procedures*
- The result of transforming an authorized state with an operation allowed in that state is an authorized state

Protection State Transitions

- Sequences of state transitions are represented by commands that update the access control matrix
- Primitive commands
  - **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*

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
```

Special Rights

- The **copy right** (or grant right) allows the possessor to grant rights to another
- The **own right** enables the possessors to add or delete privileges for themselves and others
- Principle of attenuation of Privilege
  - A subject may not give rights it does not possess to another

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

Foundational Results

Overview

- Safety Question
- HRU Model

The General Question

- Given a computer system, how can we determine if it is secure?
  - Is there a generic algorithm that allows us to determine whether a computer system is secure?
- What do we mean by “secure”?
  - Use access control matrix to express the policy

Safety

- Let *R* be the set of generic (primitive) rights of the system
  - No special rights *copy* and *own*
- Definition: when a generic right *r* is added to an element of the access control matrix not already containing *r*, that right *r* is said to be *leaked*

- Definition: If a system can never leak right *r*, the system is called safe with respect to the right *r*. If the system can leak right *r*, the system is called unsafe with respect with the right *r*

Safety vs. Security

- Safety refers to the abstract model and security refers to the actual implementation
  - A secure system corresponds to a model safe with respect to all rights
  - A model safe with respect with all rights does not ensure a secure system

Safety Question

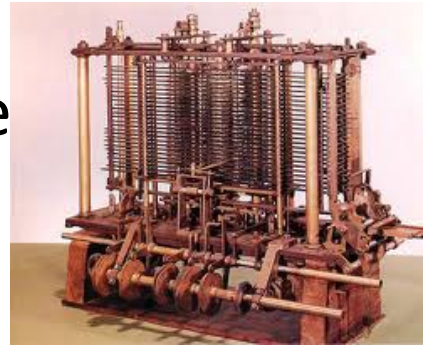
- Does there exist an algorithm for determining whether a protection system *S* with initial state s_0 is safe with respect to a generic right *r*?
  - Here, “safe” = “secure” for an abstract model

Mono-Operational Commands

- Theorem: There exists an algorithm that will determine whether a given *mono-operational* protection system with initial state s_0 is safe with respect to a generic right *r*
- Sketch of proof
  - Each command is identified by the primitive operation it invokes. Consider the minimal sequence of commands needed to leak r from the system with initial state s_0. We can show that the length of this sequence is bounded. Therefore, we can enumerate all possible states and determine wither the system is safe.

General Case

- Theorem: It is undecidable whether a given state of a given protection system is safe for a given generic right
- Proof sketch: we show that an arbitrary Turing machine can be reduced to the safety problem, with the Turing machine entering a final state corresponding to the leaking of a given generic right. Then if the safety problem is decidable, we can determine when the Turing machine halts. Since we already know that the halting problem is undecidable, the safety problem can not be decidable either.



Key Points

- Safety problem undecidable
- Limiting scope of systems can make problem decidable