

CS590U

Access Control: Theory and Practice

Lecture 6 (January 27)

The Harrison-Ruzzo-Ullman Model



Papers That This Lecture is Based Upon

- M.A. Harrison, W.L. Ruzzo, and J.D. Ullman: Protection in Operating Systems. *Communications of the ACM*, August 1976.
- M.A. Harrison and W.L. Ruzzo: Monotonic Protection Systems. In *Foundations of Secure Computation*, 1978.



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 $a(X_1, X_2, \dots, X_k)$

if

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

then

$op_1 \quad \dots \quad op_n$

end

- X_1, \dots, X_k are formal parameters



Six Primitive Operations

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



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 $\alpha(x, y, z)$
 - enter r1 into (x,x)
 - destroy subject x
 - enter r2 into (y,z)
 - end
- One can never use $\alpha(s,s,o)$ to change a state



Example 4 in [HRU]:

- **Problem:** how to Implementing Unix access control in HRU
- **Difficulty:** the owner of a file may specify the privileges of all other users
- **Solution:** the cell (f,f) determines who can access the file f
- **Question:** anything to say about this solution? other solutions?



The Safety Problem

- What do we mean by “safe”?
 - Definition 1: “access to resources without the **concurrency** 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.



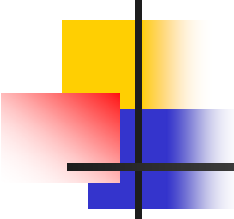
Definition of the Safety Problem in [HRU]

- 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 $\alpha(x_1, \dots, x_k)$ 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 .



Let Us Look at the Mathematical Problem

- Given a protection system, a state of the system, determines whether a right could be leaked
- Undecidable in the general case



Simulating Turing Machines using Protection Systems

- The set of generic rights include
 - the states and tape symbols of the Turing machine,
 - and two special rights: `own', `end'
- Turing Machine instructions are mapped to commands



Turing Machine

- A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$
 - Q is the set of states
 - Σ is the input alphabet
 - Γ is the tape alphabet
 - δ is the transition function
 - $q_0 \in Q$ is the start state
 - $q_{\text{accept}} \in Q$ is the accept state
 - $q_{\text{reject}} \in Q$ is the reject state, $q_{\text{reject}} \neq q_{\text{accept}}$

Mapping a Tape to an Access Matrix

- The j 'th cell on the tape = the subject s_j
- The j 'th cell has symbol $X \Rightarrow X \in (s_j, s_j)$
- The head is at the j 'th cell and the current state is $q \Rightarrow q \in (s_j, s_j)$
- The k 'th cell is the last \Rightarrow
 $\text{'end'} \in (s_k, s_k)$
- For $1 \leq j < k$, $\text{'own'} \in (s_j, s_{j+1})$

Moving Left:

$(q, X) \rightarrow (p, Y, \text{left})$

```
command  $C_{qX}(s, s')$ 
  if  $q$  in  $(s', s')$  and  $X$  in  $(s', s')$ 
    and `own' in  $(s, s')$ 
  then  delete  $q$  from  $(s', s')$ 
        delete  $X$  from  $(s', s')$ 
        enter  $Y$  into  $(s', s')$ 
        enter  $p$  into  $(s, s)$ 
  end
```

Moving Right (case one):

$(q, X) \rightarrow (p, Y, \text{right})$

command $C_{qX}(s, s')$

if q in (s, s) and X in (s, s)

and ' own' in (s, s')

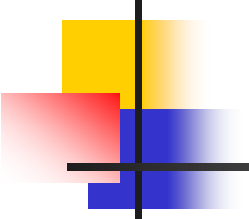
then delete q from (s, s)

delete X from (s, s)

enter Y into (s, s)

enter p into (s', s')

end



Moving Right (case two): $(q, X) \rightarrow (p, Y, \text{right})$

```
command  $C_{qX}(s, s')$ 
  if  $q$  in  $(s, s)$  and  $X$  in  $(s, s)$ 
    and `end' in  $(s, s)$ 
  then  delete  $q$  from  $(s, s)$       delete  $X$  from  $(s, s)$ 
        enter  $Y$  into  $(s, s)$ 
        create subject  $s'$       enter `own' into  $(s, s')$ 
        enter  $p$  into  $(s', s')$   enter  $B$  into  $(s', s')$ 
        delete end from  $(s, s)$   enter `end' into  $(s', s')$ 
  end
```



Summary

- Given a Turing Machine, it can be encoded as a protection system, so that the Turing Machine enters the accept state iff the HRU protection system leaks the right corresponding to q_{accept}
- Safety in HRU is thus undecidable.



Other Results

- The safety question is
 - decidable for mono-operational
 - PSPACE-complete for systems without create
 - undecidable for biconditional monotonic protection systems
 - decidable for monoconditional monotonic protection systems



The Take-Grant Model

- Two special rights `take' and `grant'
- The state is represented by a graph
- **The take rule:** if x has `take' right over z , and z has right r over y , then x can get right r over y
- **The grant rule:** if z has `grant' right over x , and z has right r over y , then x can get right r over y



The Take and the Grant Rule

- **The take rule:** if x has 'take' right over z , and z has right r over y , then x can get right r over y
- **The grant rule:** if z has 'grant' right over x , and z has right r over y , then x can get right r over y



Other Models

- Schematic Protection Model
- Typed Access Matrix Model
 - developed by Ravi Sandhu, et al.



End of Lecture 6

- Next lecture
 - HRU, safety, Take-Grant examined