CS 556 – Computer Security
Fall 2012

Dr. Indrajit Ray
Email: indrajit@cs.colostate.edu

Department of Computer Science
Colorado State University
Fort Collins, CO 80523, USA
HRU Model of Authorization System

Safety of Access Control Systems

DAC Recent Trends

DAC in Database Systems

HRU Model of Authorization System
HRU Model of Authorization Systems

- Objective
  - We want to be able to specify the dynamic propagation of access rights
  - We want to be able to analyze the protection scheme provided by the system under the dynamic nature of the system

  - Introduces the concepts of *authorization systems* and *safety* of such systems
Consists of

- An initial access matrix
- The initial state
- A finite set of HRU commands expressed as a program that operates on the access matrix
- We call this the Protection Scheme / Authorization Scheme / Protection System / Authorization System
HRU Model

- At any time the access matrix gives the set of access rights subjects have to objects in the system
- The access matrix changes by the execution of authorized HRU commands as autonomously invoked by subjects
Changing Authorization State

- Six primitive operations on access matrix
- Application of an operation \( op \) to state

\[
Q = (S, O, A)
\]

causes state transition to

\[
Q' = (S', O', A')
\]
## Changing Authorization State

<table>
<thead>
<tr>
<th>Operation</th>
<th>Condition</th>
<th>Resulting State $Q'$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>enter</strong> $r$ into $A[s_i, o_j]$</td>
<td>$s_i \in S$  \ $o_j \in O$</td>
<td>$S' = S$, $O' = O$  \ $A'[s_i, o_j] = A[s_i, o_j] \cup {r}$  \ $A'[s_k, o_l] = A[s_k, o_l]$ for $k \neq i, l \neq j$</td>
</tr>
<tr>
<td><strong>delete</strong> $r$ from $A[s_i, o_j]$</td>
<td>$s_i \in S$  \ $o_j \in O$</td>
<td>$S' = S$, $O' = O$  \ $A'[s_i, o_j] = A[s_i, o_j] \setminus {r}$  \ $A'[s_k, o_l] = A[s_k, o_l]$ for $k \neq i, l \neq j$</td>
</tr>
<tr>
<td><strong>create subject</strong> $s_i$</td>
<td>$s_i \notin O$</td>
<td>$S' = S \cup {s_i}$, $O' = O \cup {s_i}$  \ $A'[s_i, o] = A[s, o]$, $s \in S$, $o \in O$  \ $A'[s_i, o] = {}$, $o \in O'$  \ $A'[s_i, o] = {}$, $s \in S'$, $o_j = s_i$</td>
</tr>
<tr>
<td><strong>delete subject</strong> $s_i$</td>
<td>$s_i \in S$</td>
<td>$S' = S \setminus {s_i}$, $O' = O \setminus {s_i}$  \ $A'[s, o] = A[s, o]$, $s \in S'$, $o \in O'$</td>
</tr>
<tr>
<td><strong>create object</strong> $o_i$</td>
<td>$o_i \notin O$</td>
<td>$S' = S$, $O' = O \cup {o_i}$  \ $A'[s, o] = A[s, o]$, $s \in S$, $o \in O$  \ $A'[s, o] = {}$, $o = o_j$</td>
</tr>
<tr>
<td><strong>delete object</strong> $o_i$</td>
<td>$o_i \in O$</td>
<td>$S' = S$, $O' = O \setminus {o_i}$  \ $A'[s, o] = A[s, o]$, $s \in S'$, $o \in O'$</td>
</tr>
</tbody>
</table>
Command < name > (X₁, . . . , Xₙ)
if / *test condition* /
then / *sequence of primitive operations* /
    enter / delete privilege in cell
    create / delete object
    create / delete subject
    :
fi
End
HRU Characteristics

- The HRU program is executed only when the condition is satisfied
  - condition = conjunction of tests
  - test = presence of right in cell of access matrix
- HRU commands cannot test for the absence of rights
  - Mutually exclusive rights cannot be modeled where one right is the complement of the other
- Does not specify who initiates the command
- The initial access matrix can be an empty matrix
Command $COPY \rightarrow READ(U, V, C)$

if $rc \in A[U, C] \land r \in A[U, C]$ then
  enter $r$ in $A[V, C]$
fi
End
Another HRU Example

Command $\text{TRANSFER} - \text{OWNERSHIP}(U, V, C)$

if
  $\text{own} \in A[U, C] \land g \in A[U, C]$
then
  enter own in $A[V, C]$
  delete own from $A[U, C]$
fi
End
<table>
<thead>
<tr>
<th>HRU Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HRU Model of Authorization System</strong></td>
</tr>
<tr>
<td><strong>Safety of Access Control Systems</strong></td>
</tr>
<tr>
<td><strong>DAC Recent Trends</strong></td>
</tr>
<tr>
<td><strong>DAC in Database Systems</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>no restrictions</td>
</tr>
<tr>
<td>Mono-conditional</td>
<td>condition part has only one term</td>
</tr>
<tr>
<td>Bi-conditional</td>
<td>condition part has at most two terms</td>
</tr>
<tr>
<td>Mono-operational</td>
<td>command body has only one primitive operation</td>
</tr>
<tr>
<td>No creation</td>
<td>no create primitive operation in command body</td>
</tr>
<tr>
<td>No subject creation</td>
<td>objects can be created but not subjects</td>
</tr>
<tr>
<td>Monotonic</td>
<td>no delete operation</td>
</tr>
</tbody>
</table>
SAFETY OF ACCESS CONTROL SYSTEMS
Safety Problem

- An access matrix $M$ **leaks the right** $r$ if there is a command $C : M \rightarrow M'$ that adds the right $r$ in a cell of $M$ that did not previously contain $r$.
- An access matrix $M$ is **safe with respect to the access right** $r$ if no sequence of commands can transform $M$ into a state that leaks $r$. 
System State

- The *state of a system* is the collection of the current values of all memory locations, all secondary storage, and all registers and other components of the system.
- The *state of the protection system* is the subset of such a collection that deals with allocation of access permissions; it is thus presented by the access control matrix.
HRU Model – Safety of States

- Definition 1: “access to resources without the concurrence of the owner is impossible” [HRU76]
- 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” [HRU76]
Given an access matrix $M$ and a right $r$, verifying the safety of $M$ with respect to $r$ is an undecidable problem.
Example of “Unsafe” Protection

Command $grant_{execute}(s, p, f)$
if
  $o \in A[s, f]$ then
    enter $x$ in $A[p, f]$
fi
End

Command $modify_{own\_right}(s, f)$
if
  $x \in A[s, f]$ then
    enter $w$ in $A[s, f]$
fi
End
Example (cont’d)

- Suppose user Bob has developed an application program; he wants this program to be run by other users but not modified by them.
- The previous protection system is not safe with respect to this policy; consider the following sequence of commands:
  - Bob: grant_execute(Bob, Tom, P1)
  - Tom: modify_own_right(Tom, P1)

It results in an access matrix with \( w \in A[\text{Tom}, P1] \)
<table>
<thead>
<tr>
<th>Safety of Access Control Systems</th>
<th>HRU Model of Authorization System</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Mono-conditional</td>
<td>Decidable</td>
</tr>
<tr>
<td>Bi-conditional</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Mono-operational</td>
<td>Decidable, NP complete</td>
</tr>
<tr>
<td>No creation</td>
<td>Decidable, Pspace complete</td>
</tr>
<tr>
<td>No subject creation</td>
<td>Decidable, Exponential space</td>
</tr>
<tr>
<td>Monotonic</td>
<td>Not known</td>
</tr>
</tbody>
</table>
Security is hopeless

- Not quite so; we do have safety results for interesting classes of systems

We are not interested in general classes of systems but only in a few specific systems

- Even for a specific system we need to analyze it under a variety of assumptions about behavior of individual subjects
Typed Access Matrix (TAM) Model

- Much like HRU except
  - All subjects and objects are strongly typed
  - Commands have typed parameters
  - The initiator of a TAM command is specified
    - It is assumed that whatever is listed as the first parameter, is the one to execute the TAM command
TAM Program

Command $< name > (X_1 : t_1, \ldots, X_n : t_n)$
if / * test condition * /
p $\in A[X_i, X_j] \land q \in A[X_k, X_l] \land \ldots$
then / * sequence of primitive operations */
enter / delete privilege in cell
create / delete object
create / delete subject
:
fi
End
TAM versus HRU

- TAM has
  - Stronger safety properties than HRU
  - Ease of expressiveness by means of strong typing
DAC Recent Trends
DAC - Recent Trends

- DAC models have been widely investigated in the database area
  - First model by Griffiths and Wide
- Flexibility enhanced by various extension
  - Positive versus negative authorization
  - Strong versus weak authorization
  - Implicit versus explicit
  - Content based authorization
Positive vs Negative Authorization

- Positive permissions – Grant access
- Negative permission – Deny access
- Useful for specifying stricter control for crucial data items
- Problem with conflicts between permissions
Authorization Conflicts

- Ideally make sure no conflicts
- Negative permissions take precedence
- Positive permissions take precedence
- Most specific permissions take precedence
Implicit vs Explicit Permissions

- Implicit permissions can be derived
  - by a set of propagation rules exploiting subject, object and privilege hierarchies
  - by a set of user specified derivation rules
Content-based Authorization

- Content-based access control conditions the access to a given object based on its content
- Example – A subject can access information about employees whose salaries do not exceed $30,000.
DAC IN DATABASE SYSTEMS
Database Access Control

- DBMS can provide access control for database
  - Assume authenticated users

- DBMS provides specific access rights to portions of the database
  - E.g. create, insert, delete, update, read, write
  - To entire database, tables, selected rows or columns
  - Possibly dependent on contents of a table entry
Can support a range of policies

- Centralized administration: A small number of privileged users may grant and revoke access rights
- Ownership-based administration: The owner (creator) of a table may grant and revoke access rights to the table
- Decentralized administration: In addition to granting and revoking access rights to a table, the owner may grant and revoke authorization to other users, allowing them to grant and revoke access rights to the table
Security Through Views

- View: A virtual table defined by an SQL query
- Can be used to customize relations and to provide security
- Views restricts user to only limited information from the database
  - The DBMS transforms the queries on views into queries on the base tables
### Example Base Table

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPT</th>
<th>SALARY</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Toy</td>
<td>10,000</td>
<td>Jones</td>
</tr>
<tr>
<td>Jones</td>
<td>Toy</td>
<td>15,000</td>
<td>Baker</td>
</tr>
<tr>
<td>Baker</td>
<td>Admin</td>
<td>40,000</td>
<td>Harding</td>
</tr>
<tr>
<td>Adams</td>
<td>Candy</td>
<td>20,000</td>
<td>Harding</td>
</tr>
<tr>
<td>Harding</td>
<td>Admin</td>
<td>50,000</td>
<td>None</td>
</tr>
</tbody>
</table>
Example View

CREATE VIEW TOY-DEPT
AS SELECT NAME, SALARY, MANAGER
FROM EMP
WHERE DEPT = 'TOY'

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>10,000</td>
<td>Jones</td>
</tr>
<tr>
<td>Jones</td>
<td>15,000</td>
<td>Baker</td>
</tr>
</tbody>
</table>
Example View (continued)

CREATE VIEW TOY-EMP-MGR
AS SELECT NAME, MANAGER
FROM TOY-DEPT

<table>
<thead>
<tr>
<th>NAME</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Jones</td>
</tr>
<tr>
<td>Jones</td>
<td>Baker</td>
</tr>
</tbody>
</table>
Query Modification

Alice:

GRANT SELECT ON EMP to THOMAS
WHERE SALARY < 15000

Thomas:

SELECT * FROM EMP

DBMS:

SELECT * FROM EMP
WHERE SALARY < 15000
Stored Procedures

- Some systems allow for compiling a program first and then executing it later. The user who compiles a program becomes the owner of the program, and gives other execute privilege using the RUN command.
- GRANT RUN ON program-TEST TO ALICE
- Suppose program-TEST needs to access the relation EMP. ALICE can execute program-TEST though she does not have permission to access EMP.
The GRANT Command

GRANT <privilege | role> ON <relation>
TO <users | role | PUBLIC>
[WITH GRANT OPTION]

- GRANT SELECT ON EMP TO ALICE
- GRANT SELECT ON EMP TO BOB WITH GRANT OPTION
- GRANT SELECT, UPDATE(SALARY) ON EMP TO JIM, JILL
- GRANT ALL PRIVILEGES ON EMP TO SMITH
- GRANT SELECT(NAME, DEPT) ON EMP TO PUBLIC

- The GRANT command applies to base relations as well as views
The REVOKE Command

REVOKE <privileges | role> [ON <relations>]
FROM <users | role | PUBLIC>

- REVOKE SELECT ON EMP FROM ALICE
- REVOKE UPDATE ON EMP FROM SMITH
- REVOKE DBA FROM ABRAMS
Cascading Authorizations – GRANT

HRU MODEL OF AUTHORIZATION SYSTEM

SAFETY OF ACCESS CONTROL SYSTEMS

DAC RECENT TRENDS

DAC IN DATABASE SYSTEMS

Dr. Indrajit Ray, Computer Science Department
Cascading Authorizations – REVOKE

HRU Model of Authorization System

Safety of Access Control Systems

DAC Recent Trends

DAC in Database Systems