DAC – The HRU Model

- The Harrison-Ruzzo-Ullman (HRU) has introduced some important concepts:
  - The notion of *authorization systems*
  - The notion of *safety*

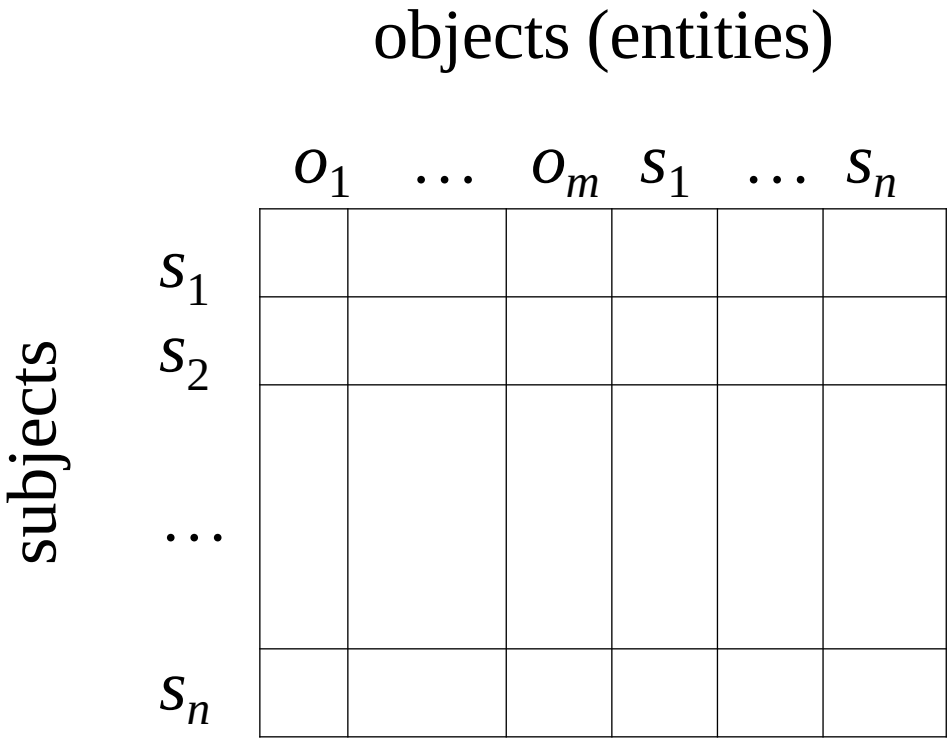
[HRU76] M.Harrison, W. Ruzzo, J. Ullman. Protection in Operating Systems. *Comm. of ACM* 19(8), August 1976.

The HRU Model

To describe the HRU model we need:

- S be a set of subjects
- O be a set of objects
- R be a set of access rights
- an access matrix $M = (M_{so})_{s \in S,\, o \in O}$
- the entry M_{so} is the subset R specifying the rights subject s has on object o

Access Restriction Facility



- 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

Access Control - basic concepts

	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

Access Control - basic concepts

EXAMPLE: The UNIX system defines the rights "read," "write," and "execute." When a process accesses a file, these terms mean what one would expect. When a process accesses a directory, "read" means to be able to list the contents of the directory; "write" means to be able to create, rename, or delete files or subdirectories in that directory; and "execute" means to be able to access files or subdirectories in that directory. When a process accesses another process, "read" means to be able to receive signals, "write" means to be able to send signals, and "execute" means to be able to execute the process as a subprocess.

Protection State Transitions

- State $X_i = (S_i, O_i, A_i)$
- Transitions τ_i
  - Single transition $X_i \vdash_{\tau_{i+1}} X_{i+1}$
  - Series of transitions $X \vdash^* Y$
- Access control matrix may change
  - Change command c associated with transition
  - $X_i \vdash_{c_{i+1}(p_{i+1},\ldots,p_{i+1})} X_{i+1}$
- Commands often called *transformation procedures*

Special Privileges: Copy, Ownership

- Copy (or grant)
  - Possessor can extend privileges to another
- Own right
  - Possessor can change their own privileges
- Principle of Attenuation of Privilege
  - A subject may not give rights it does not possess

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.

The HRU Model – Primitive Operations

The model includes six *primitive operations* for manipulating the set of subjects, the set of objects, and the access matrix:

- **enter** *r* into M_{so}
- **delete** *r* from M_{so}
- **create subject** *s*
- **delete subject** *s*
- **create object** *o*
- **delete object** *o*

The HRU Model - Commands

Commands in the HRU model have the format

command $c(x_1,.....,x_k)$
if r_1 in $M_{s1,o1}$ **and**
if r_2 in $M_{s2,o2}$ **and**
:
if r_m in $M_{sm,om}$
then $op_1,.....,op_n$
end

The HRU Model - Commands

- The indices s_1,.....,s_m and o_1,.....,o_m are subjects and objects that appear in the parameter list $c(x_1,.....,x_k)$
- The condition part of the command checks whether particular access rights are present; the list of conditions can be empty
- If all conditions hold, then the sequence of basic operations is executed
- Each command contains at least one operation
- Commands containing exactly one operation are said *mono-operational* commands

The HRU Model – Command examples

command create_file (*s*,*f*)
    create *f*
    enter o into $M_{s,f}$
    enter r into $M_{s,f}$
    enter w into $M_{s,f}$
end

command grant_read (*s*,*p*,*f*)
    if o in $M_{s,f}$
    then enter r into $M_{p,f}$
end

Primitive Commands

Suppose the process p wishes to create a new process q. The following command would capture the resulting changes in the access control matrix.

```
command spawn•process(p, q)
    create subject q;
    enter own into a[p, q];
    enter r into a[p, q];
    enter w into a[p, q];
    enter r into a[q, p];
    enter w into a[q, p];
end
```

Mono-operational Command

EXAMPLE: The command

command *make•owner*(p, f)
    enter *own* **into** $a[p, f]$;
end

Conditional Command

command $grant\bullet read\bullet file\bullet 1(p, f, q)$
    if own **in** $a[p, f]$
    then
        enter r **into** $a[q, f]$;
end

Multiple Conditional

```
command grant•read•file•2(p, f, q)
    if r in a[p, f] and c in a[p, f]
    then
        enter r into a[q, f];
end
```

Multiple Conditional

```
if own in a[p, f] or a in a[p, f]
then
    enter r into a[q, f];

command grant•write•file•1(p, f, q)
    if own in a[p, f]
    then
        enter r into a[q, f];
end
command grant•write•file•2(p, f, q)
    if a in a[p, f]
    then
        enter r into a[q, f];
end
```

grant•write•file•1(p, f, q); grant•write•file•2(p, f, q);

The HRU Model – Protection Systems

- A protection system is defined as
  - A finite set of rights
  - A finite set of commands
- A protection system is a state-transition system

The HRU Model - States

- The effects of a command are recorded as a change to the access matrix (usually the modified access control matrix is denoted by *M*')
- Hence the access matrix describes the state of the *protection system*
- What do we mean by the state of the protection system?
  - 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

The HRU Model – States

Definition. A state, i.e. an access matrix *M*, is said to *leak* the right *r* if there exists a command *c* that adds the right *r* into an entry in the access matrix that previously did not contain *r*. More formally, there exist *s* and *o* such that $r \notin M_{so}$ and, after the execution of *c*, $r \in M'_{so}$.

Note: The fact that an right is leaked is not necessarily bad; many systems allow subjects to give other subjects access rights

The HRU Model – Safety of States

What do we mean by saying that a state is “safe”?

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]

The HRU Model – Safety

The problem motivating the introduction of safety can be described as follows:

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

The HRU Model – An example of “unsafe” protection system

Assume to have a protection system with the following two commands:

command grant_execute (*s*,*p*,*f*)
    if o in $M_{s,f}$
    then enter x into $M_{p,f}$
end

command modify_own_right (*s*,*f*)
    if x in $M_{s,f}$
    then enter w into $M_{s,f}$
end

The HRU Model – An example of “unsafe” protection system

- 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 access matrix where the entry $M_{\text{Tom,P1}}$ contains the w access right

The HRU Model - Safety

Definition. Given a protection system and a 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*.

Alternative (more intuitive) definition. A state of a protection system, that is, its matrix *M*, is said to be safe with respect to the right *r* if no sequence of commands can transform *M* into a state that leaks *r*.

Theorem. Given an access matrix *M* and a right *r*, verifying the safety of *M* with respect to *r* is an undecidable problem.

The HRU Model – Safety
Other relevant results

The safety question is

- decidable for mono-operational protection systems
- undecidable for biconditional monotonic protection systems
  - Monotonic protections system means deletion of access rights are not allowed once it is entered in the protection system.
  - Biconditional means there is exactly two conditions in the precondition part of the commands.
- decidable for monoconditional monotonic protection systems
  - Monoconditional means there is exactly one condition in the precondition part of the commands.

The HRU Model
Concluding Remarks

The results on the decidability of the safety problem illustrate an important security principle, the *principle of economy of mechanisms*

- if one designs complex systems that can only be described by complex models, it becomes difficult to find proofs of security
- in the worst case (undecidability), there does not exist a universal algorithm that verifies security for all problem instances

Other Theoretical Models

- The take-grant model
  (by A. Jones, R. Lipton, and L. Snyder)
- The schematic protection model
  (by R. Sandhu)
- The typed access matrix model
  (by R. Sandhu)

Other Models

- DAC models have been widely investigated in the area of DBMS
- The first DAC model for relational databases has been developed by Griffiths and Wide
- Several extensions to such model have been developed

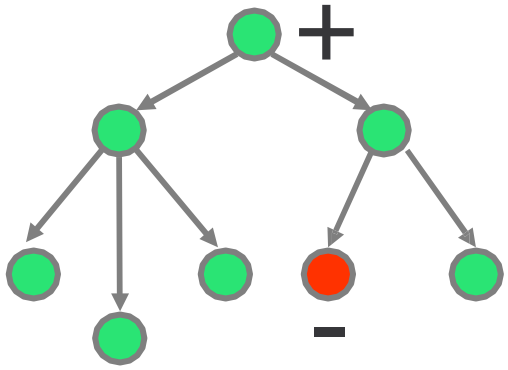
DAC – additional features and recent trends

- Flexibility is enhanced by supporting different kinds of permissions
  - Positive vs. negative
  - Strong vs. weak
  - Implicit vs. explicit
  - Content-based

Positive and Negative Permissions

- Positive permissions ➔ Give access
- Negative permissions ➔ Deny access
- Useful to specify exceptions to a given policy and to enforce stricter control on particular crucial data items

Positive and Negative Permissions



Main Issue: Conflicts

Authorization Conflicts

- Main solutions:
  - Negative permissions take precedence
  - Positive permissions take precedence
  - Most specific permissions take precedence

Weak and Strong Permissions

- Strong permissions cannot be overwritten
- Weak permissions can be overwritten by strong and weak permissions

Implicit and Explicit Permissions

- Some models support implicit permissions
- Implicit permissions can be derived:
  - by a set of *propagation rules* exploiting the subject, object, and privilege hierarchies
  - by a set of user-defined *derivation rules*

Derivation Rules: Example

- Ann can read file F1 from a table if Bob has an explicit denial for this access
- Tom has on file F2 all the permissions that Bob has
- Derivation rules are a way to concisely express a set of security requirements

Derivation Rules

- Derivation rules are often expressed according to logic programming
- Several research efforts have been carried out to compare the expressive power of such languages
- We need languages based on SQL and/or XML

Content-based Permissions

- Content-based access control conditions the access to a given object based on its content
- This type of permissions are mainly relevant for database systems
- As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K

Content-based Permissions

- Two are the most common approaches to enforce content-based access control in a DBMS:
  - by associating a predicate (or a Boolean combination of predicates) with the permission
  - by defining a *view* which selects the objects whose content satisfies a given condition, and then granting the permission on the view instead of on the basic objects