

# Chapter 2 ACCESS CONTROL MATRIX

## 2.1 PROTECTION STATE

- **Protection system:** describes the conditions under which a system is secure.
- **Access control matrix:** has arisen in operating systems and database research.
- **State of a system:** the collection of the current values of all
  - memory locations
  - secondary storage
  - registers and other components of the system
- **Protection state:** the subset of the above collection that deals with protection.
- **Access control matrix:** a tool that can describe the current protection state.
- **Protection states:**
  - ***P***: The set of possible protection states.
  - ***Q***: a subset of ***P***. Consists of exactly those states in which the system is authorized to reside.
  - When the system is in state ***Q***, the system is secure.
  - When the system is in state ***P-Q***, the system is not secure.

# Chapter 2 ACCESS CONTROL MATRIX

- **Security policy:** characterization of the states in  $Q$ .
- **Security mechanism:** prevents the system from entering a state in  $P$ - $Q$ .
- **Access control matrix:**
  - characterizes the rights of each subject with respect to every other entity.
  - The description of elements of  $A$  form a specification (against which the current state can be compared with).
  - As the system changes, the protection state changes.
  - When a command changes the state of the system, a state transition occurs.
  - In practice, any operation on a real system causes multiple state transitions.
    - Reading, loading, altering, and execution of any datum or instruction causes a transition.
    - Only those state transitions that affect the protection state of the system are considered.

# Chapter 2 ACCESS CONTROL MATRIX

## 2.2 ACCESS CONTROL MATRIX MODEL

- **Access control matrix:**
  - describes the rights of users over files in a matrix.
  - Butler Lampson proposed this model in 1971.
  - Graham and Denning refined it. Their version will be used in this chapter.
- **Set of objects  $O$ :** set of all protected entities.
- **Set of subjects  $S$ :** set of active objects (e.g., processes and users).
- **Set of rights  $R$ :** rights associated with entities.
- **Example:**  $a[s,o]$ ,  $s \in S$ ,  $o \in O$ , and  $a[s,o] \subseteq R$ .
- **$(S, O, A)$ :** set of protection states of the system.

# Chapter 2 ACCESS CONTROL MATRIX

objects (entities)

	$o_1$	...	$o_m$	$s_1$	...	$s_n$
$s_1$						
$s_2$						
...						
$s_n$						

subjects

- Subjects  $S = \{ s_1, \dots, s_n \}$
- Objects  $O = \{ o_1, \dots, o_m \}$
- Rights  $R = \{ r_1, \dots, r_k \}$
- Entries  $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \dots, r_y \}$   
means subject  $s_i$  has rights  $r_x, \dots, r_y$  over object  $o_j$

# Chapter 2 ACCESS CONTROL MATRIX

	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

Figure 2-1 An access control matrix with 2 processes and 2 files.

- The set of rights is {read, write, execute, append, own}
- In the above matrix:
  - Process 1 can read or write file 1, and can read file 2.
  - Process 2 can append to file 1 and read file 2.
  - Process 1 can communicate with process 2 by writing to it.
  - Process 2 can read from process 1.

# Chapter 2 ACCESS CONTROL MATRIX

- Each process owns itself and the file with the same #.
- The processes are treated as both subjects (rows) and objects (columns). This allows a process to be the target of operations as well as the operators.
- **EXAMPLE:**
  - The UNIX systems defines the rights “read,” “write,” and “execute.”
  - When a process accesses a file, these terms are used in an expected way.
  - 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 the directory.
    - “execute” means to be able to access files or subdirectories in the directory.

# Chapter 2 ACCESS CONTROL MATRIX

- **When a process accesses another process:**
  - “read” means to be able to receive signals.
  - “write” means to be able to send signals.
  - “execute” means to be able to execute the process as a subprocess.
- **The superuser can access any local file regardless of the permission the owner has granted. However, the superuser cannot alter a directory using the system calls and commands that alter files.**

# Chapter 2 ACCESS CONTROL MATRIX

- Although the objects associated with an access control matrix are normally files, devices, and processes, they could also be messages sent between processes, or systems themselves.

Host names	<i>telegraph</i>	<i>nob</i>	<i>toadflax</i>
<i>telegraph</i>	own	ftp	ftp
<i>nob</i>		ftp, nfs, mail, own	ftp, nfs, mail
<i>toadflax</i>		ftp, mail	ftp, nfs, mail, own

Figure 2-2 Rights on a LAN. The set of rights is {ftp, mail, nfs, own}

- The rights correspond to various network protocols:
  - own: the ability to add servers.
  - ftp: the ability to access the system using the File Transfer Protocol (FTP)
  - nfs: the ability to access file systems using the Network File System (NFS) protocol
  - mail: the ability to send and receive mail using the Simple Mail Transfer Protocol (SMTP)

## Chapter 2 ACCESS CONTROL MATRIX

- ***telegraph:***
  - a personal computer with an *ftp* client but no servers.
  - neither of the other systems can access it but it can *ftp* to them.
- ***nob:***
  - provides NFS service to a set of clients that does not include the host *toadflax*.
  - both systems can exchange *mail* with any host and allow any host to use *ftp*.

## Chapter 2 ACCESS CONTROL MATRIX

- At the microlevel, access control matrices can model programming languages. In this case, the *objects* are variables and the subjects are the *procedures* (or modules).
- Consider a program in which events must be synchronized. A module provides functions for:
  - incrementing (*inc\_ctr*)
  - decrementing (*dec\_ctr*)
  - the routine *manager* calls these functions.
  - “+” and “-“ are the rights, representing the ability to add and subtract, respectively.
  - “call” is the ability to invoke a procedure.
  - The routine *manager* can call itself. It may be recursive.

	<i>counter</i>	<i>inc_ctr</i>	<i>dec_ctr</i>	<i>manager</i>
<i>inc_ctr</i>	+			
<i>dec_ctr</i>	-			
<i>manager</i>		call	call	call

Figure 2-3 Rights in a program. The set of rights is {+,-,call}.

# Chapter 2 ACCESS CONTROL MATRIX

## 2.3 PROTECTION STATE TRANSITIONS

- When processes execute operations, the state of the protection systems changes.
- Initial state of the system:  $X_0 = (S_0, O_0, A_0)$ .
- The set of state transitions:  $T_1, T_2, \dots$
- The successive states:  $X_1, X_2, \dots$
- $X_i \xrightarrow{\tau_{i+1}} X_{i+1}$ : state transition  $\tau_{i+1}$  moves the system from state  $X_i$  to state  $X_{i+1}$ .
- $X_i \xrightarrow{*} Y$ : a system starts at some state  $X$ , and enters state  $Y$  after a series of transitions.
- The representation of the protection system as an access control matrix should also be updated.
- Let  $ck$  be the  $k$ th command with formal parameters  $p_{k,1}, \dots, p_{k,m}$ . The  $i$ th transition is:  $X_i \xrightarrow{ci+1(p_{i+1,1}, \dots, p_{i+1,m})} X_{i+1}$ .

## Chapter 2 ACCESS CONTROL MATRIX

- A set of *primitive commands* that alter the access control matrix:
  - create subject  $s$ : creates a new subject  $s$ . This operation does not add any rights.
  - create object  $o$ : creates a new object  $o$ . This operation does not add any rights.
  - enter  $r$  into  $a[s,o]$ : adds the right  $r$  to the cell  $a[s,o]$ .
  - delete  $r$  from  $a[s,o]$ : deletes the right  $r$  from the cell  $a[s,o]$ .
  - destroy subject  $s$ : deletes the subject  $s$ . The column and row for  $s$  in  $A$  are also deleted.
  - destroy object  $o$ : deletes the object  $o$ . The column for  $o$  in  $A$  is also deleted.
- These primitive commands can be combined into commands for which *multiple* commands may be executed.

## Chapter 2 ACCESS CONTROL MATRIX

- **EXAMPLE:** Process  $p$  creates file  $f$  with owner read ( $r$ ) and write ( $w$ ) 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
```

- The system can update the matrix only by using defined commands; it cannot use the primitive commands directly.
- **EXAMPLE:** The following command is mono-operational. It does not delete any existing owner rights; it just adds  $p$  to the set of owners of  $f$ .

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

# Chapter 2 ACCESS CONTROL MATRIX

- 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
```

- Any number of conditions may be placed together using “and.” Let  $p$  give  $q$   $r$  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  $a[q, f]$ ;
  end
```

## Chapter 2 ACCESS CONTROL MATRIX

- **Monoconditional commands:**
  - commands with one condition.
  - The command *grant•read•file•1* is monoconditional.
- **Biconditional commands:**
  - commands with two conditions.
  - The command *grant•read•file•2* is biconditional.
- **As both commands have one primitive command, they are mono-operational.**

# Chapter 2 ACCESS CONTROL MATRIX

- All conditions are joined by “and,” and never by “or.” Joining commands with “or” is equivalent to two commands, each with one condition.

- **EXAMPLE:**

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

is equivalent to

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

and

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

- These two commands can be written as follows:  

```
grant•read•file•1(p,f,q); grant•read•file•2(p,f,q);
```
- The negation of a command is not permitted. One cannot test for the absence of a right within a command by the condition:  

```
if r not in a[p,f];
```