These Slides are prepared from
Matt Bishop slides and book "Introduction to Computer Security"
Benefiting from the Slides posted by Ahmad Al-Mulhem

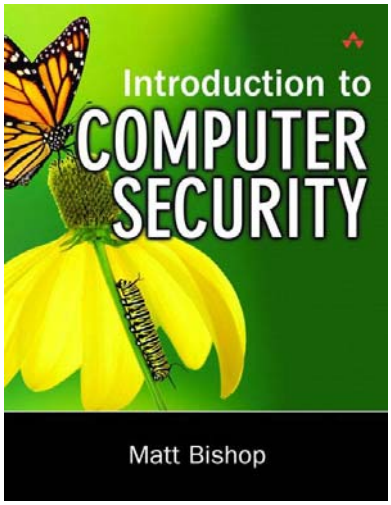
Access Control

Access Control Matrix ***Ch 2***

Access Control Mechanisms Ch 14

Adnan Gutub
gutub@kfupm.edu.sa

Computer Engineering Department
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia



Ch 2: Access Control Matrix

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

Protection State

- The "state" of a system is the collection of the current values of all memory locations (RAM, registers, secondary storage, etc.)
- The "protection state" of a system is the subset of states that deals with protection
  - Describes current settings, values of system relevant to protection
- Access control matrix model - tool
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

Access Control

- subjects (e.g. users or processes) ➔ objects (e.g. files)
  - Subjects (active) need to access objects (passive)
- Two main categories:
  - Discretionary access control (DAC):
    - defined by user ------- Discretionary = optional
  - Mandatory access control (MAC):
    - enforced by the system

Access Control Matrix (ACM)

The set of all protected entities is the set *O*

Processes and users constitute the set *S*

objects (entities)

subjects

	o_1	…	o_m	s_1	…	s_n
s_1						
s_2						
…						
s_n						

- 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: read, write, execute, append, own

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

Example 2

- Processes: telegraph, nob, toadflax
- Rights: own, ftp, nfs, mail

host names	*telegraph*	*nob*	*toadflax*
telegraph	own	ftp	ftp
nob		ftp, nfs, mail, own	ftp, nfs, mail
toadflax		ftp, mail	ftp, nfs, mail, own

Example 3

- Procedures: inc_ctr, dec_ctr, manage
- Variable: counter
- Rights: +, –, call

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

State Transitions

- As processes execute, the protection state of system changes
  - Based on history, time, day, functions, rights of other subjects…etc
  - Conditions (no access to file while being in writing function) …etc
- The initial state of the system $X_0 = (S_0, O_0, A_0)$
  - State Transitions set of operations is represented as: $\tau_1, \tau_2, \ldots$
  - Successive states $X_0, X_1, X_2, \ldots$
- $\vdash$ represents transition
  - $X_i \vdash_{\tau_{i+1}} X_{i+1}$: command τ_{i+1} moves system from state X_i to X_{i+1}
  - $X_i \vdash^{*} X_{??}$: a sequence of commands moves system from state X_i to $X_{??}$
- Commands often called *transformation procedures*

Primitive Operation Commands

- **create subject** *s*
  - Creates new row, column in ACM
- **create object** *o*
  - Creates new column in 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*
- **destroy subject** *s*
  - Deletes row, column from ACM
- **destroy object** *o*
  - Deletes column from ACM

Multi-Operational Commands 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

Mono-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 & Mono-operational command
  - Single condition & single primitive operation in this command

Multi-Conditions Commands

- 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
```

Commands with 2 conditions = bi-conditional
Commands with 2 primitive commands = bi-operational

Conditions are joined by `and` only

- All multi-conditions are joined by `and` and never `or` !! Why ??
- Also, conditions cannot be based on negation such as:

```
if r not in A[p, f]    xxx
```


Key Points

- Access control matrix (ACM) is a simple theoretical (not practical) tool for representing protection state
- Transitions - commands- alter protection state
- 6 primitive operations change matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions

These Slides are prepared from
Matt Bishop slides and book "Introduction to Computer Security"
Benefiting from the Slides posted by Ahmad Al-Mulhem

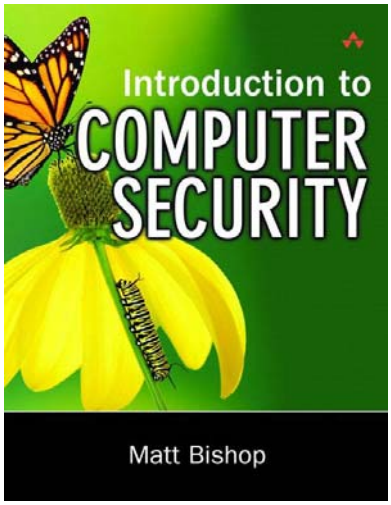
Access Control

Access Control Matrix Ch 2

***Access Control Mechanisms* Ch 14**

Adnan Gutub
gutub@kfupm.edu.sa

Computer Engineering Department
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia



Chapter 14: Access Control Mechanisms

- Access control lists
- Capabilities
- Locks and keys
- Ring-based access control
- Propagated access control lists

Problems with Access control matrix:

- number of subjects and objects can be very large
- most entries are blank/empty
- creations and deletions are expensive

Access Control Lists (ACL)

- ACL = Columns of "access control matrix"

		file1	*file2*	*file3*
Andy	rx	r	rwo	
Betty	rwxo	r		
Charlie	rx	rwo	w	

- ACLs:
- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

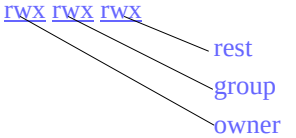
Default Permissions

- Normal: if not named, *no* rights over file
  - Principle of Fail-Safe Defaults
- If many subjects, may use groups or wildcards in ACL

- Example (UNICOS)
  - entries are (*user*, *group*, *rights*)
    - If *user* is in *group*, has rights over file
    - '*' is wildcard for *user*, *group*
      - (holly, *, r): holly can read file regardless of her group
      - (*, gleep, w): anyone in group gleep can write file

ACL Abbreviations

- ACLs can be long … so combine users
  - Example (UNIX): 3 classes of users: owner, group, rest
  - rwx rwx rwx



  - Separate read, write, execute rights for user, group and rest
  - Ownership assigned based on creating process
    - "-" means no right (e.g. rw- r- - r- -)

Abbreviated ACL + Full ACL

- Problem: Abbreviated ACL results in loss of granularity
- Solution: Augment abbreviated lists with full ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies

- Example: IBM AIX
  - Base permissions are abbreviated ACL, extended permissions are ACLs with user, group
  - ACL entries can add rights, but on deny, access is denied

Creation and Maintenance of ACLs

- Which subjects can modify an object's ACL?
  - creator is given own right that allows this
- Are there privileged users (root)? Do ACLs apply to them?
- Does the ACL support groups or wildcards?
- How are conflicts in access control permissions handled?

Capability Lists (C-List)

- C-List: Rows of access control matrix

		file1	*file2*	*file3*
Andy	rx	r	rwo	
Betty	rwxo	r		
Charlie	rx	rwo	w	

- C-Lists:
- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

C-List Meaning

- Like a bus ticket
  - Mere possession indicates rights that subject has over object
  - Object identified by capability (as part of the token)
    - Name may be a reference, location, or something else
  - Architectural construct in capability-based addressing; this just focuses on protection aspects
- Must prevent process from altering capabilities
  - Otherwise subject could change rights encoded in capability or object to which they refer
- C-List typically supported by hardware

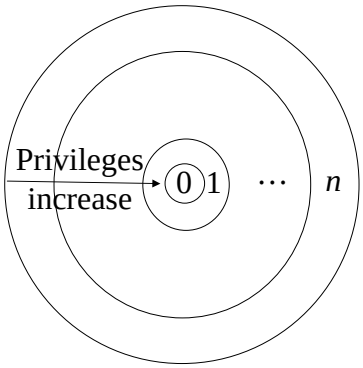
ACLs vs. Capabilities

- Both theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
  - ACLs answer second easily; C-Lists, first
- Suggested that the second question, which in the past has been of most interest, is the reason ACL-based systems more common than capability-based systems
  - As first question becomes more important (in incident response, for example), this may change

Locks and Keys

- Associate information (*lock*) with object, information (*key*) with subject
  - Latter controls what the subject can access and how
  - Subject presents key; if it corresponds to any of the locks on the object, access granted
- This can be dynamic
  - ACLs, C-Lists static and must be manually changed
  - Locks and keys can change based on system constraints, other factors (not necessarily manual)

Ring-Based Access Control



- n layers of privilege (rings)
  - ring 0 is most privileged
  - ring n is least privileged
- Introduced by Multics (1964-2000)
- Typically hardware enforced
- Special gates between rings are provided to allow an outer ring to access an inner ring's resources

Key Points

- Access control matrix (ACM) is a simple tool to model protection state but:
  - most entries are blank/empty
  - expensive to manage (creations, modification and deletions)
- Access control mechanisms provide controls for users accessing files & resources
- Many different forms
  - ACLs, capabilities, locks and keys
  - Ring-based mechanisms (Mandatory)