COP 4610

Operating System Principles

Protection

1

Overview

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

2

Objectives

- Discuss the **goals and principles** of **protection** in a modern computer system
- Explain how protection domains combined with an **access matrix** are used to specify the resources a process may access
- Examine **capability-based** protection systems

3

Goals of Protection

- In one protection model, a computer consists of a **collection** of objects, hardware or software
- Each object has a **unique name** and can be accessed through a **well-defined set of operations**
- **Protection problem** - ensure that each object is accessed **correctly** and only by those processes that are **allowed** to do so

4

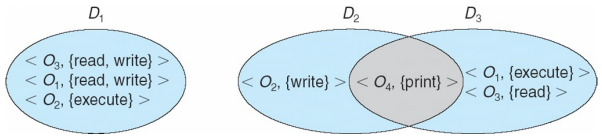
Principles of Protection

- Guiding principle – **principle of least privilege**
  - Static
  - Dynamic - **domain switching, privilege escalation**
  - "Need to know" a similar concept regarding access to data
  - "Containment of failure"
- Must consider "grain" aspect
  - Rough-grained
  - Fine-grained
- Domain can be user, process, procedure

5

Domain Structure

- Access-right = <*object-name*, *rights-set*> where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights



6

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed (in a similar fashion "setgid")
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

7

Password Example

```
cooluser@LAPTOP-5V55HON5:~$ ls -l /etc/shadow
-rw-r—— 1 root shadow 1824 Oct 18 19:49 /etc/shadow
cooluser@LAPTOP-5V55HON5:~$
```

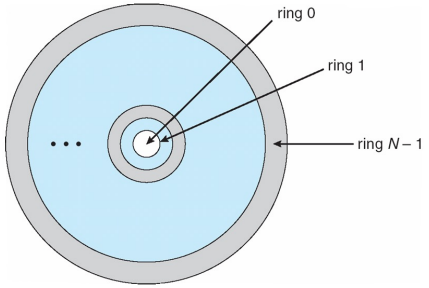
```
cooluser@LAPTOP-5V55HON5:~$ passwd
Changing password for cooluser.
Current password:
New password:
Retype new password:
passwd: password updated successfully
cooluser@LAPTOP-5V55HON5:~1$
```

```
cooluser@LAPTOP-5V55HON5:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 142144 Sep  5  2019 /bin/ls
cooluser@LAPTOP-5V55HON5:~$ ls -l /bin/passwd
-rwsr-xr-x 1 root root 68208 May 28 01:37 /bin/passwd
cooluser@LAPTOP-5V55HON5:~$
```

8

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < i \Rightarrow D_i \subseteq D_j$



Multics Benefits and Limits

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex -> more overhead
- But does not allow strict need-to-know
  - Object accessible in D_j but not in D_i, then j must be $< i$
  - But then every segment accessible in D_i also accessible in D_j

Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- *Access(i, j)* is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

11

Access Matrix

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

12

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who **creates object** can define access column for that object
- Can be expanded to dynamic protection
  - Operations to **add, delete** access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* D_i *to* D_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

13

Access Matrix Example

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

14

Access Matrix Example

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix Example

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix Example

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode

Implementation of Access Matrix

- Generally, a **sparse matrix**
- **Option 1** – Global table
  - Store ordered triples < *domain, object, rights-set* > in table
  - A requested operation M on object O_j within domain D_i -> search table for < D_i, O_j, R_k >
    - with M ∈ R_k
  - But table could be very large
  - Difficult to group objects (e.g., an object that all domains can read)
- **Option 2** – Access lists for objects
  - Each column implemented as an access list for a specific object (and **stored** with said object)
  - Resulting **per-object list** consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access

19

Implementation of Access Matrix

- **Option 3** - Capability list for domains
  - Instead of object-based, list is domain-based
  - **Capability list** for domain is list of objects together with operations allowed on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed

20

Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
- Most systems use **combination of access lists and capabilities**
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed

Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent