Operating Systems

20. Protection

Paul Krzyzanowski
Rutgers University
Spring 2015

Protection & Security

- Security
  - Prevention of unauthorized access to a system
    - Prevent malicious or accidental access
    - "access" may be:
      - user login, a process accessing things it shouldn't, physical access
    - The access operations may be reading, destruction, or alteration
- Protection
  - The mechanism that provides and enforces controlled access of resources to processes
  - A protection mechanism *enforces* security policies

Principle of Least Privilege

At each abstraction layer, every element (user, process, function) should be able to access ***only*** the resources necessary to perform its task

- Even if an element is compromised, the scope of damage is limited
- Consider:
  - Good: You cannot kill another user's process
  - Good: You cannot open the /etc/hosts file for writing
  - Good: Private member functions & local variables in functions limit scope
  - Violation: a compromised print daemon allows someone to add users
  - Violation: a process can write a file even though there is no need to
  - Violation: admin privileges set by default for any user account
- Least privilege is often difficult to define & enforce

Privilege Separation

Divide a program into multiple parts: high & low privilege components

- Example on POSIX systems
  - Each process has a *real* and *effective* user ID
  - Privileges are evaluated based on the effective user ID
    - Normally, `uid == euid`
  - An executable file may be tagged with a *setuid bit*
    - **`chmod +sx filename`**
    - When run: uid = user's ID
      euid = file owner's ID (without setuid, runs with user's ID)
  - Separating a program
    1. Run a *setuid* program
    2. Create a communication link to self (*pipe*, *socket*, shared memory)
    3. *fork*
    4. One of the processes will call `seteuid(getuid())` to lower its privilege

Security Goals

- Authentication
  - Ensure that users, machines, programs, and resources are properly identified
- Integrity
  - Verify that data has not been compromised: deleted, modified, added
- Confidentiality
  - Prevent unauthorized access to data
- Availability
  - Ensure that the system is accessible

The Operating System

The OS provides processes with access to resources

Resource	OS component
Processor(s)	Process scheduler
Memory	Memory Management + MMU
Peripheral devices	Device drivers & buffer cache
Logical persistent data	File systems
Communication networks	Sockets

- Resource access attempts go through the OS
- OS decides whether access should be granted
  - Rules that guide the decision = *policy*

Domains of protection

- Processes interact with objects
  - Objects include:
    hardware (CPU, memory, I/O devices)
    software: files, processes, semaphores, messages, signals
- A process should be allowed to access only objects that it is authorized to access
  - A process operates in a **protection domain**
  - Protection domain defines the objects the process may access and how it may access them

7

Modeling Protection: Access Matrix

Rows: domains

Columns: objects

Each entry represents an access right of a domain on an object

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer
D_0	read	read-write	print
D_1	read-write-execute	read	
D_2	read-execute		
D_3		read	print
D_4			print

8

Access Matrix: Domain Transfers

Switching from one domain to another is a configurable policy

A process in D_0 can switch to running in domain D_1

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read	read-write	print	–	switch	switch		
D_1	read-write-execute	read			–			
D_2	read-execute				switch	–		
D_3		read	print					
D_4			print					

9

Access Matrix: Additional operations

Copy: allow delegation of rights

- Copy a specific access right on an object from one domain to another
  - Rights may specify either a copy or a transfer of rights

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read	read-write	print	–	switch	sw		
D_1	read-write-execute	read*						
D_2	read-execute				swtich	–		
D_3		read	print					
D_4			print					

A process executing in D_1 can give a read right on F_1 to another domain

10

Access Matrix: Additional operations

Owner: allow new rights to be added or removed

- An object may be identified as being *owned* by the domain
- Owner can add and remove any right in any column of the object

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read owner	read-	print	–	switch	sw		
D_1	read-write-execute	read*			–			
D_2	read-execute				swtich			
D_3		read	print					
D_4			print					

A process executing in D_0 can give a read right on F_0 to domain D_3 and remove the execute right from D_1

11

Access Matrix: Additional operations

Control: change entries in a row

- If *access(i, j)* includes a *control* right, then a process executing in Domain *i* can change access rights for Domain *j*

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read owner	read-write	print	–	switch	swtich		
D_1	read-write-execute	read*			–			control
D_2	read-execute				swtich	–		
D_3		read	print					
D_4			print					

A process executing in D_1 can modify any rights in domain D_4

12

Implementing an access matrix

- A single table is usually impractical
  - Big size: # domains (users) × # objects (files)
  - Objects may come and go frequently
- Access Control List
  - Associate a column of the table with each object

13

Implementing an access matrix

- Access Control List
  - Associate a column of the table with each object

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read owner	read-write	[illegible]	[illegible]	[illegible]	[illegible]		
D_1	read-write-execute	read*			–			
D_2	read-execute				swtich	–		
D_3		read	print					
D_4			print					

ACL for file F_0

14

Example: Limited ACLs in POSIX systems

Problem: an ACL takes up a varying amount of space (possibly a lot!)
- Won't fit in an inode

UNIX Compromise:
- A file defines access rights for three domains:
  - the owner, the group, and everyone else
- Permissions
  - Read, write, execute, directory search
  - Set user ID on execution
  - Set group ID on execution
- Default permissions set by the *umask* system call
- *chown* system call changes the object's owner
- *chmod* system call changes the object's permissions

15

Example: Full ACLs in POSIX systems

- *What if we really want a full ACL?*
- Extended attributes: stored outside of the inode
  - Hold an ACL
  - And other name:value attributes
- Enumerated list of permissions on users and groups
  - Operations on all objects:
    - *delete, readattr, writeattr, readextattr, writeextattr, readsecurity, writesecurity, chown*
  - Operations on directories
    - *list, search, add_file, add_subdirectory, delete_child*
  - Operations on files
    - *read, write, append, execute*
  - Inheritance controls

16

Implementing an access matrix

Capability List
- Associate a row of the table with each domain

objects (columns) / *domains of protection* (rows)

	F_0	F_1	Printer	D_0	D_1	D_2	D_3	D_4
D_0	read owner	read-write	print	–	switch	swtich		
D_1	read-write-execute	read*			–			
D_2	read-execute				swtich	–		
D_3		read	print					
D_4			print					

Capability list for domain D_1

17

Capability Lists

- List of objects together with the operations allowed on the objects
- Each item in the list is a *capability*: the operations allowed on a specific object
- A process presents the capability along with a request
  - Possessing the capability means that access is allowed
- A process cannot modify its capability list

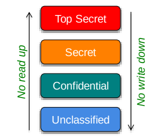
18

Access Control Models: MAC vs. DAC

- DAC: Discretionary Access Control
  - A subject (domain) can pass information onto any other subject
  - In some cases, access rights may be transferred
  - *Most systems use this*
- MAC: Mandatory Access Control
  - Policy is centrally controlled
  - Users cannot override the policy

Multi-level Access Control

- Typical MAC implementations use a Multi-Level Secure (MLS) access model
- Bell-LaPadula model
  - Identifies the ability to access and communicate data
  - Objects are classified into a hierarchy of sensitivity levels
    - Unclassified, Confidential, Secret, Top Secret
  - Each user is assigned a clearance
  - "No read up; no write down"
    - Cannot read from a higher clearance level
    - Cannot write to a lower clearance level
- Works well for government information
- Does not translate well to civilian life



Confidential cannot read Secret
Confidential cannot write Unclassified

The End