Operating Systems Design
19. Protection

Paul Krzyzanowski
pxk@cs.rutgers.edu

Protection & Security

- Security
  - Prevention of unauthorized access to a system
    - 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:
  - Violation: a compromised print daemon allows one 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
  - Good: Private member functions & Local variables in functions limit scope
- 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
  - Separating a program
    - Run a *setuid* program
    - Create a communication link to self (*pipe*, *socket*, shared memory)
    - *fork*
    - One process will call seteuid(getuid()) to lower its privilege

Security Goals

- Authentication
  - Ensure that users, machines, programs, and resources are properly identified
- Confidentiality
  - Prevent unauthorized access to data
- Integrity
  - Verify that data has not been compromised: deleted, modified, added
- 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:
    hardware (CPU, memory, I/O devices)
    software: files, 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

Modeling Protection: Access Matrix

- Rows: domains
- Columns: objects
- Each entry represents an access right of a domain on an object

objects

domains of protection

	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

Access Matrix: Domain Transfers

- Switching from one domain to another is a configurable policy

objects / *domains of protection*

	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					

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

domains of protection

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

E.g., a process executing in D_1 can give a read right on F_1 to domain D_2

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

domains of protection

	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					

E.g., 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

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

domains of protection

	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					

E.g., a process executing in D_1 can modify any rights in domain D_4

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

Implementing an access matrix

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

objects

domains of protection

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

ACL for file F_0

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

Example: Full ACLs in POSIX systems

- *What if we really want an ACL?*
- Extended attributes: stored outside of the inode
- 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

Implementing an access matrix

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

objects

domains of protection

	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

Capability Lists

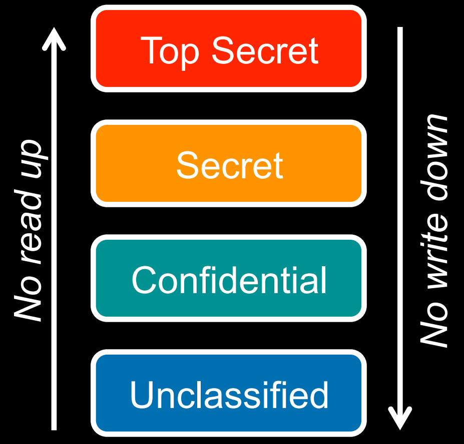
- List of objects together with the operations allowed on the objects
- Each item in the list is a *capability*: the operations allowed on a specific object
- A process presents the capability along with a request
  - Possessing the capability means that access is allowed
- A process cannot modify its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

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
  - Users are 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

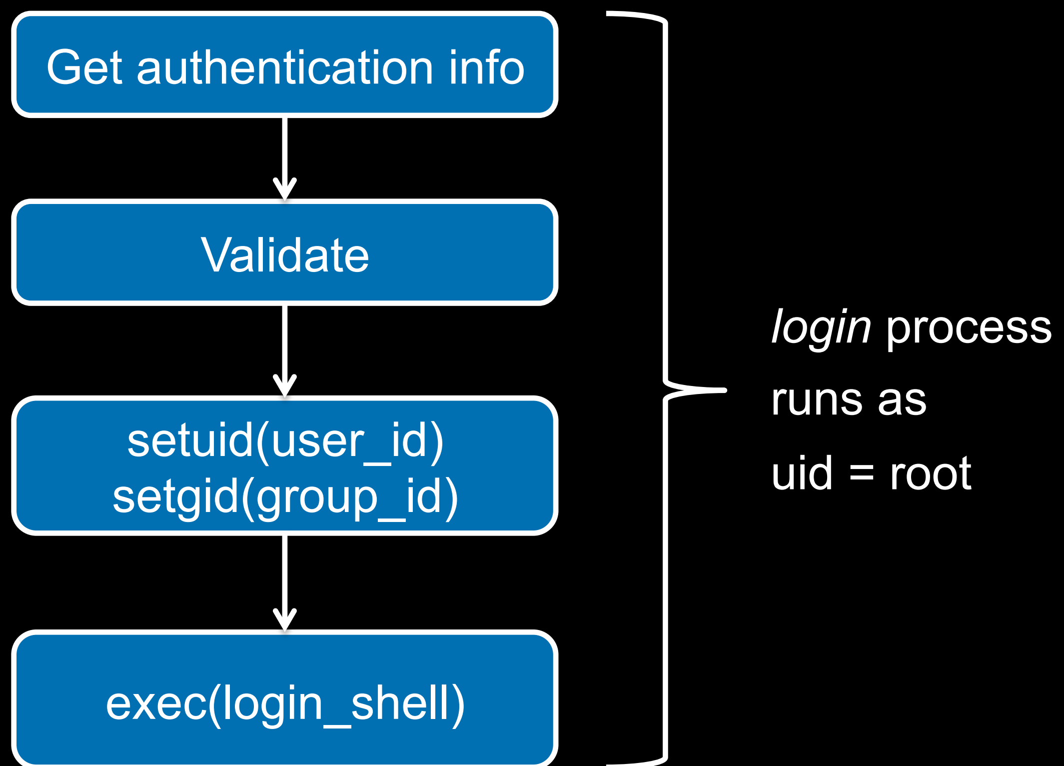


Confidential cannot read Secret
Confidential cannot write Unclassified

Authentication

Authentication

- Establish & verify identity
  - Then decide whether to allow access to resources
- Example: login



Password Authentication Protocol (PAP)

- Reusable passwords
- Server keeps a database of *username:password* mappings
- Prompt client/user for a login name & password
- To authenticate, use the login name as a key to look up the corresponding password in a database (file) to authenticate

```
if (supplied_password == retrieved_password)
      then user is authenticated
```

PAP: Reusable passwords

One problem: what if the password file isn't sufficiently protected and an intruder gets hold of it, he gets all the passwords!

Enhancement:

Store a hash of the password in a file

- given a file, you don't get the passwords
- have to resort to a dictionary or brute-force attack

- Unix approach
  - Password encrypted with 3DES hashes; then MD5 hashes; now SHA512 hashes
  - Salt used to guard against dictionary attacks

Authentication

Three factors:

- something you have *key, card*
  - can be stolen
- something you know *passwords*
  - can be guessed, shared, stolen
- something you are *biometrics*
  - costly, can be copied (sometimes)

Authentication

factors may be combined

- ATM machine: 2-factor authentication
  - ATM card *something you have*
  - PIN *something you know*

Identification vs. Authentication

- Identification:
  - Who are you?
  - User name, account number, …
- Authentication:
  - Prove it!
  - Password, PIN, encrypt nonce, …

Versus Authorization

Authorization defines access control

Once we know a user's identity:

- Allow/disallow request
- Operating system enforces system access based on user's credentials
  - Network services usually run in another context
  - Network server may not know of the user
  - Application takes responsibility
- May contact an authorization server
  - Trusted third party that will grant credentials
  - Kerberos ticket granting service
  - RADIUS (centralized authentication/authorization)

Three (Four?) A's of Security

- Authentication
  - Validate an identity or a message
- Authorization (Access Control)
  - Enforce policy
- Accounting
- Auditing

Accounting

If security has been compromised

… *what happened?*

… *who did it?*

… *how did they do it?*

Log transactions

- Logins
- Commands
- Database operations
- *Who looks at audits?*

Log to remote systems

- Minimize chances for intruders to delete logs

Auditing

Go through software source code and search for security holes

- Need access to source
  - Some operating systems > 50 million lines!
- Experienced staff + time
- E.g., OpenBSD

Complex systems will have more bugs

- And will be harder to audit

The End