

## 22C:50 Section 1 Security

Introduction to Systems Software  
Department of Computer Science  
Fall, 2002 – William F. Decker

11 December 2002

1

## Protection

- Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- Required are:
  - Method for specification of the controls to impose
  - A means of enforcement
- Protection ≠ security! Why?

11 December 2002

2

## Policy and Mechanism Separation

- Policy determines what must be done.
- A mechanism determines how the policy will be enforced.
- For purposes of flexibility:
  - General mechanisms are most desirable
  - Such mechanisms may be able to enforce a variety of possible policy choices
  - Otherwise a mechanism must be changed for each environment in which it is used

11 December 2002

3

## Minix Policy/Mechanism Example

- Policy:
  - Arbitrary programs must be able to determine the identities of users who are authorized to use the system, including their UIDs, GIDs, and home directories.
  - However, their encrypted passwords must not be exposed.

11 December 2002

4

## Minix Policy/Mechanism Example

- Mechanism:
  - Passwords are not kept in `/etc/passwd`, the file that defines users.
  - Passwords are kept in `/etc/shadow`.
  - Both files are owned by root.
  - `/etc/passwd` has `-rw-r--r--` access rights
  - `/etc/shadow` has `-rw-----` access rights

11 December 2002

5

## Domains of Protection

- Consider a computer system as a collection of processes and objects.
- “Objects” includes hardware, software, and data objects.
- Objects have unique names and can be accessed only through well-defined operations, which are usually object-specific.

11 December 2002

6

## Domains of Protection

- A process should be allowed to access only those resources for which it has authorization.
- At any given moment, a process should be able to access only those resources that it currently requires to complete its task (need-to-know principle).

11 December 2002

7

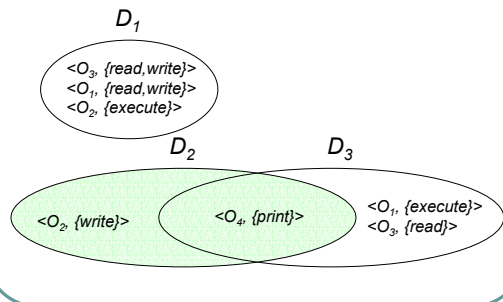
## Domains of Protection

- Protection domains:
  - Specify the objects that a process may access
  - Domains define object sets and the types of operations permitted on the objects (access rights)
  - Domains may be thought of as sets of ordered pairs:  $\langle \text{object name}, \text{rights set} \rangle$
  - Example:  $\langle \text{fileF}, \{\text{read}, \text{write}\} \rangle$
  - Domains need not be disjoint

11 December 2002

8

## Domain Structure



11 December 2002

9

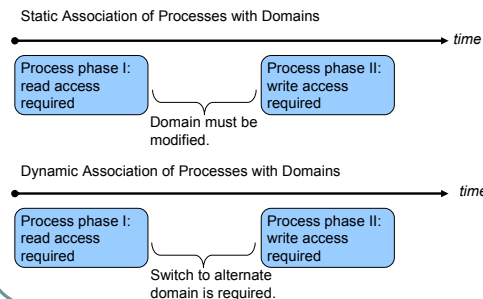
## Domains of Protection

- Associations between processes and domains can be static or dynamic.
- If static and we want to adhere to need-to-know, processes may have different requirements during their lifetime, and this would require dynamic domain modification.
- If dynamic, then switching domains can create the desired effect.

11 December 2002

10

## Example



11 December 2002

11

## Domain Realization

- Users as domains: access depends on identity of the user; domain switching occurs usually when one user logs out and another logs in
- Processes as domains: access depends on identity of process; domain switching occurs via IPC to a server process or by role playing (e.g., setuid features of Minix)
- Procedures as domains: objects that can be accessed are defined by variables defined within the procedure; domain switching occurs when a procedure call occurs

11 December 2002

12

## Minix/Unix Examples

- Domains are associated with users and groups
- Domain switching is done by temporarily changing the user identification
- Each file has an associated owner identification and a domain bit (setuid bit)
- If user A executes a file owned by B, the process executes as:
  - User A if the file's setuid bit is 0
  - User B if the file's setuid bit is 1
  - At exit, user A's identity is restored

11 December 2002

13

## Visualizing Protection as a Matrix

Domain	Object							
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

11 December 2002

14

## Adding Domain Switching to Matrix

Domain	Object											
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3	
1	Read	Read Write									Enter	
2			Read	Read Write Execute	Read Write		Write					
3						Read Write Execute	Write	Write				

11 December 2002

15

## Realization as ACLs or CLs

### Access Control Lists (ACLs):

- A condensed form of the columns of the matrix
- Each file has a list of elements of the form (UID, GID, rwx permissions)
- Minix/Unix does this in a limited, less general way

File0: (Jan,\*,RWX)

File1: (Jan,system,RWX)

File2: (Jan,\*,RW-), (Bob,staff,RW-), (Tom,\*,RW)

File3: (\*,student,R-)

File4: (Jim,\*,-),(\*,student,R-)

### Capability Lists (CLs):

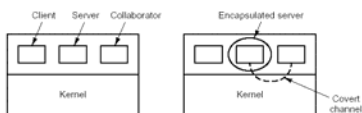
- A condensed form of the row information in the matrix
- CL is associated with a process running in a specified domain

#	Type	Rights	Object
0	File	R--	Pointer to File3
1	File	RWX	Pointer to File4
2	File	RW-	Pointer to File5
3	Pointer	--W-	Pointer to Printer1

11 December 2002

16

## Covert Channels



- Client and server do not trust each other
- Collaborator is conspiring with server
- How to keep server from leaking information to collaborator (confinement problem)?
- Probably can guard against overt leaks, using protection schemes.
- Collaborator might still monitor server behavior and derive information from that behavior.

11 December 2002

17