

CS419 – Spring 2010

Computer Security

Access Control: Policies and Mechanisms

Vinod Ganapathy

Lectures 9 and 10

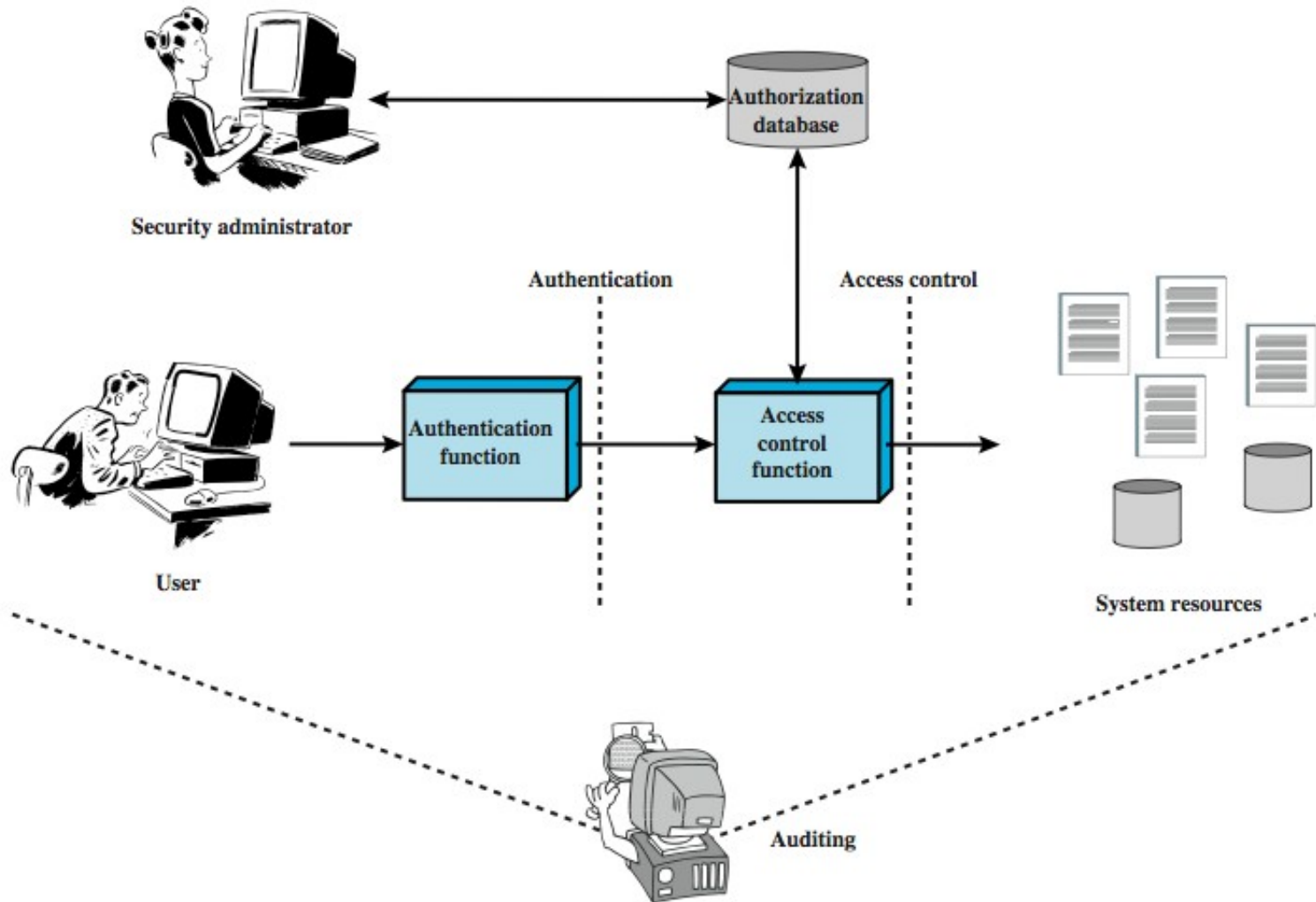
Access Control

- “The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner“
- central element of computer security
- assume have users and groups
 - authenticate to system
 - assigned access rights to certain resources on system

Access Control

- **Policy**: Decides which **subject** can perform what **operations** on which **object**
- **Mechanism**: Set of techniques used to enforce the policy

Access Control Principles



Access Control Requirements

- reliable input
- fine and coarse specifications
- least privilege
- separation of duty
- open and closed policies
- policy combinations, conflict resolution
- administrative policies

Access Control Elements

- subject - entity that can access objects
 - a process representing user/application
 - often have 3 classes: owner, group, world
- object - access controlled resource
 - e.g. files, directories, records, programs etc
 - number/type depend on environment
- access right - way in which subject accesses an object
 - e.g. read, write, execute, delete, create, search

Access Control: Overview

- Protection state of system
 - Describes current settings, values of system relevant to protection
- Access control matrix
 - Describes protection state precisely
 - Matrix describing rights of subjects
 - State transitions change elements of matrix

Discretionary Access Control

- often provided using an access matrix
 - lists subjects in one dimension (rows)
 - lists objects in the other dimension (columns)
 - each entry specifies access rights of the specified subject to that object

Description

objects (entities)

	o_1	...	o_m	s_1	...	s_n
s_1						
s_2						
...						
s_n						

subjects

- Subjects $S = \{ s_1, \dots, s_n \}$
- Objects $O = \{ o_1, \dots, o_m \}$
- Rights $R = \{ r_1, \dots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \dots, r_y \}$
means subject s_i has
rights r_x, \dots, r_y over
object o_j

Example 1

- Processes p, q
- Files f, g
- Rights r, w, x, a, o

$f \quad g \quad p \quad q$

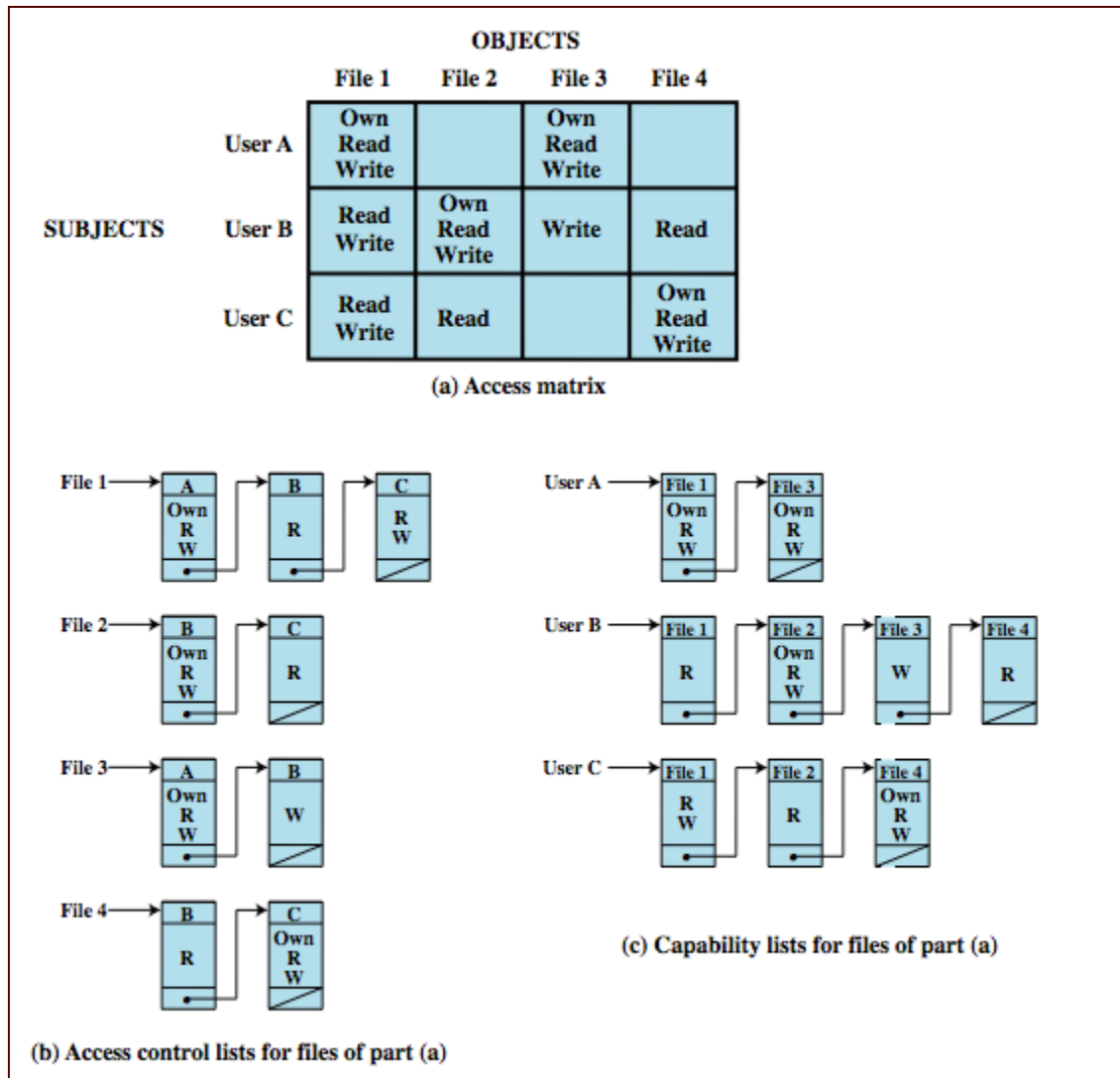
$p \quad rwo \quad r \quad rwxow$

$q \quad a \quad ro \quad r \quad rwxo$

Access control structures

- access matrix is often sparse
- can decompose by either row or column
- Two implementations, depending on how you decompose:
 - Access Control Lists
 - Capabilities

Access Control Structures

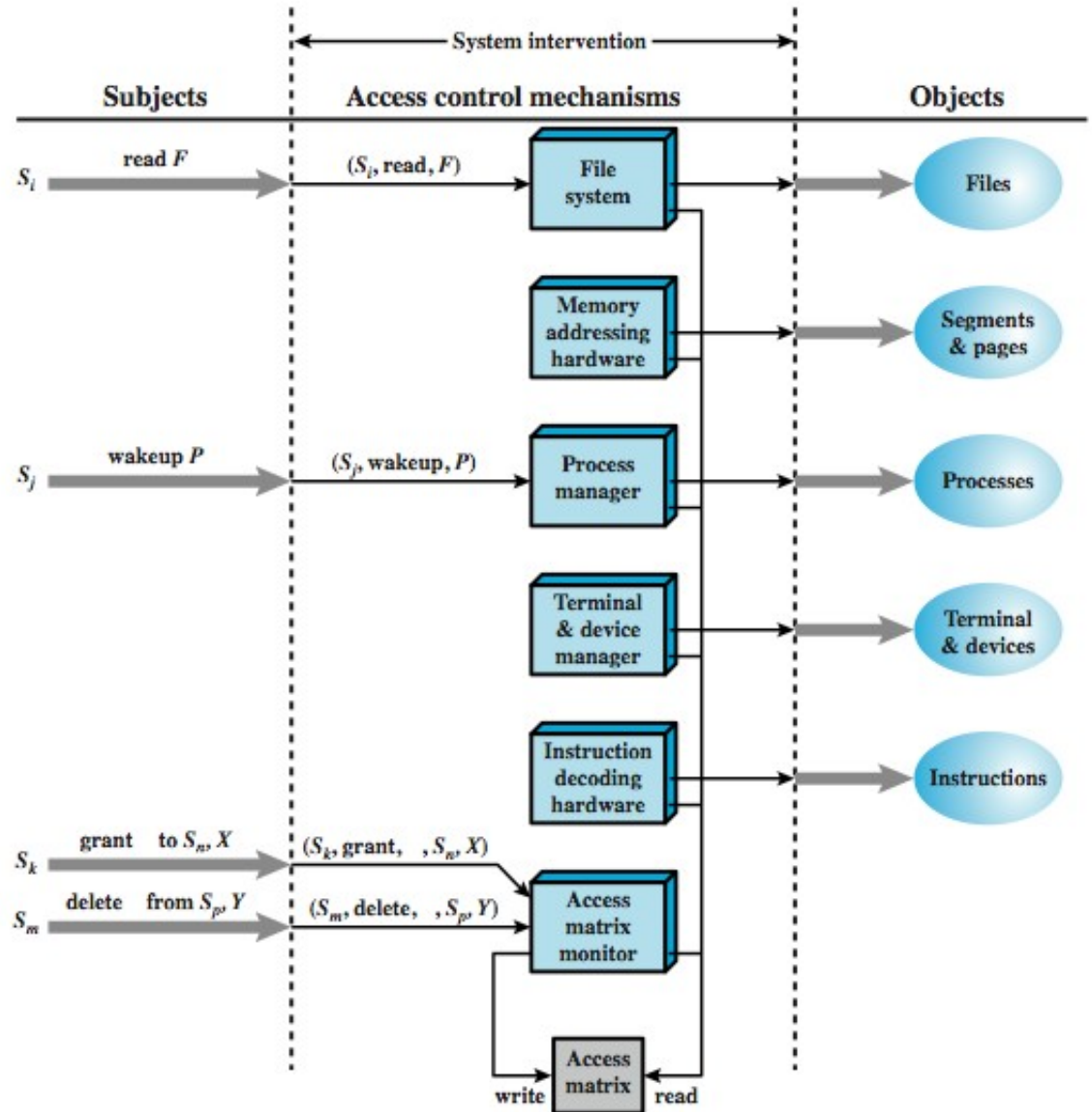


Access Control Model

		OBJECTS								
		subjects			files		processes		disk drives	
		S ₁	S ₂	S ₃	F ₁	F ₁	P ₁	P ₂	D ₁	D ₂
SUBJECTS	S ₁	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S ₂		control		write *	execute			owner	seek *
	S ₃			control		write	stop			

* - copy flag set

Access Control Function



Primitive Operations

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

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

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 command
 - Single condition in this command

Multiple Conditions

- 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
```

Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- Primitive operations alter matrix
 - Transitions can be expressed as commands composed of these operations and, possibly, conditions

What Is “Secure”?

- Adding a generic right r where there was not one is “leaking”
- If a system S , beginning in initial state s_0 , cannot leak right r , it is *safe with respect to the right r* .

Safety Question

- Does there exist an algorithm for determining whether a protection system S with initial state s_0 is safe with respect to a generic right r ?
 - Here, “safe” = “secure” for an abstract model
- Answer: **No**. Seminal result due to Harrison, Ruzzo and Ullman (1976).

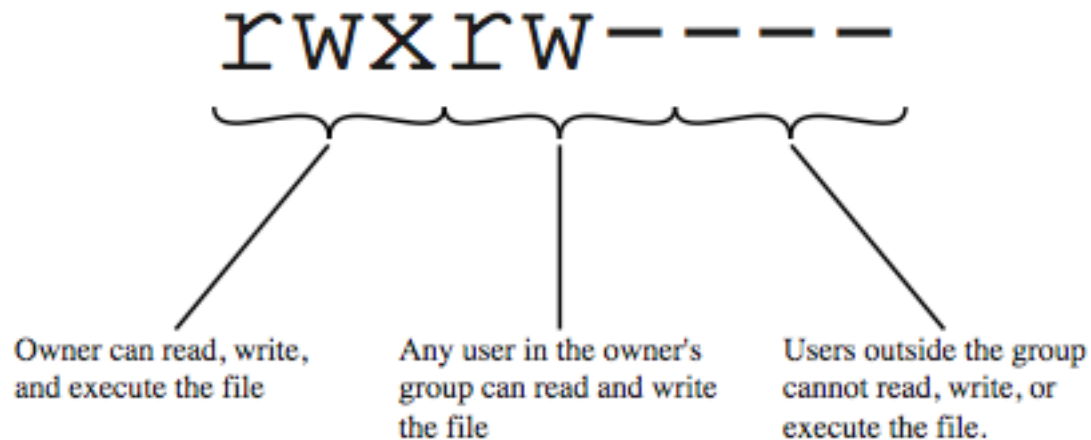
Protection Domains

- set of objects with associated access rights
- in access matrix view, each row defines a protection domain
 - but not necessarily just a user
 - may be a limited subset of user's rights
 - applied to a more restricted process
- may be static or dynamic

UNIX File Concepts

- UNIX files administered using inodes
 - control structure with key info on file
 - attributes, permissions of a single file
 - may have several names for same inode
 - have inode table / list for all files on a disk
 - copied to memory when disk mounted
- directories form a hierarchical tree
 - may contain files or other directories
 - are a file of names and inode numbers

UNIX File Access Control



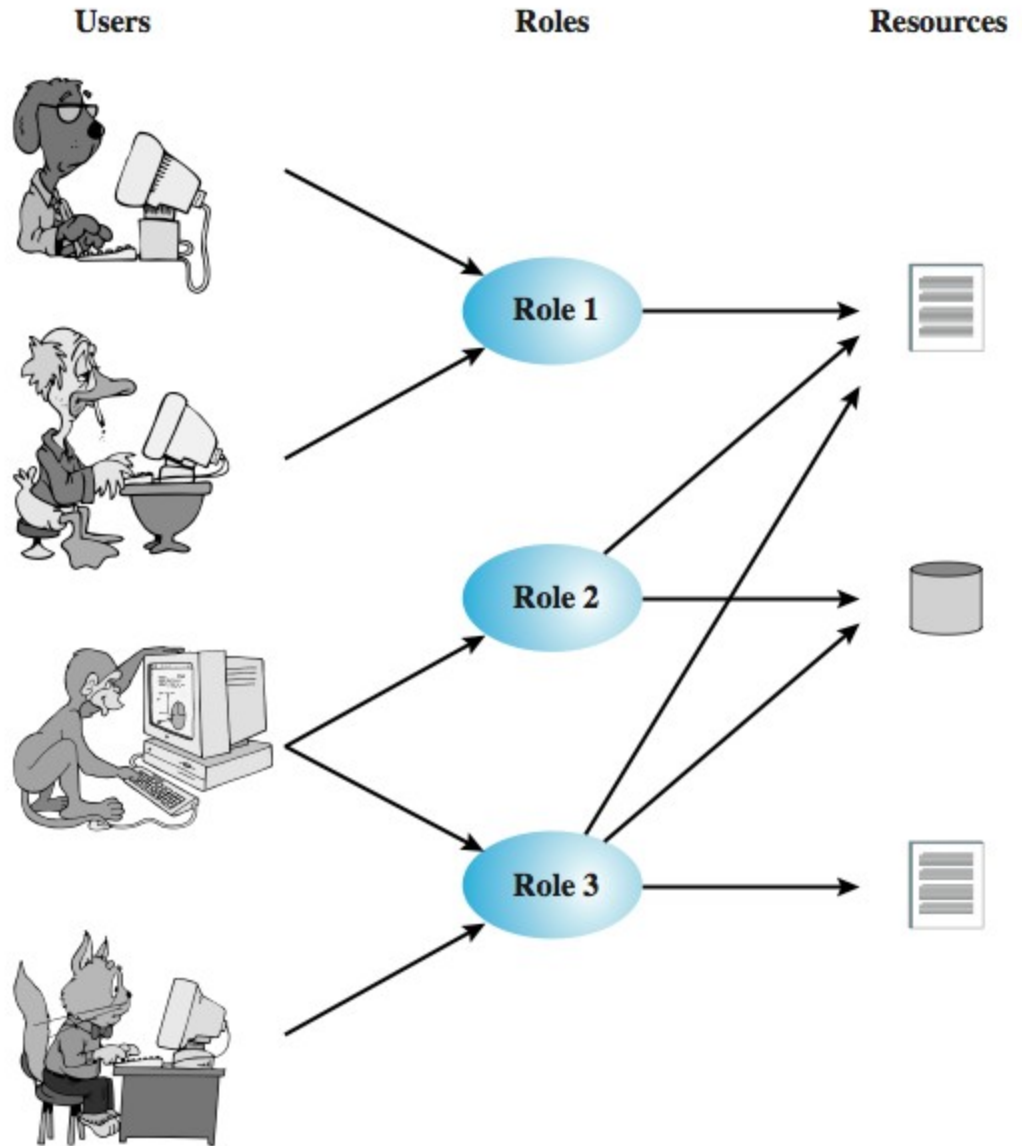
UNIX File Access Control

- “set user ID”(SetUID) or “set group ID”(SetGID)
 - system temporarily uses rights of the file owner / group in addition to the real user’s rights when making access control decisions
 - enables privileged programs to access files / resources not generally accessible
- sticky bit
 - on directory limits rename/move/delete to owner
- superuser
 - is exempt from usual access control restrictions

UNIX Access Control Lists

- modern UNIX systems support ACLs
- can specify any number of additional users / groups and associated rwx permissions
- ACLs are optional extensions to std perms
- group perms also set max ACL perms
- when access is required
 - select most appropriate ACL
 - owner, named users, owning / named groups, others
 - check if have sufficient permissions for access

Role-Based Access Control

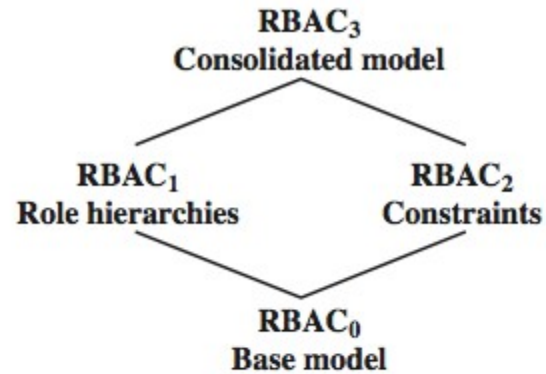


Role-Based Access Control

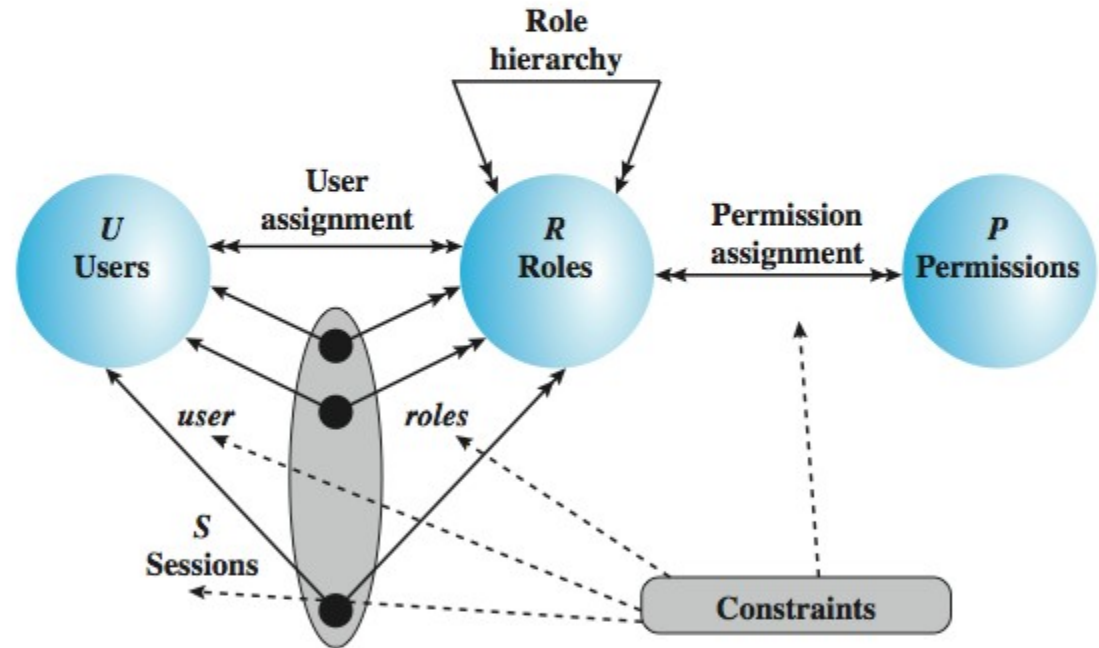
	R_1	R_2	...	R_n
U_1	×			
U_2	×			
U_3		×		×
U_4				×
U_5				×
U_6				×
•				
•				
U_m	×			

	OBJECTS								
	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
•									
•									
R_n			control		write	stop			

Role-Based Access Control

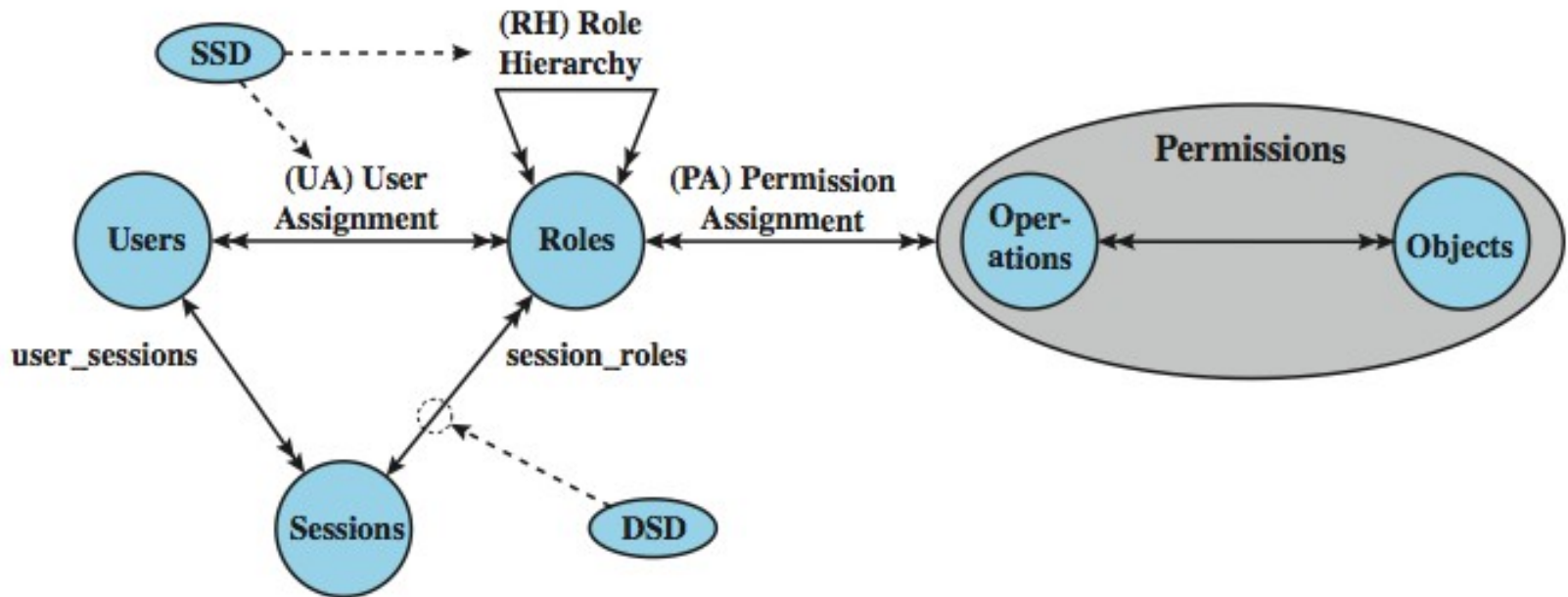


(a) Relationship among RBAC models



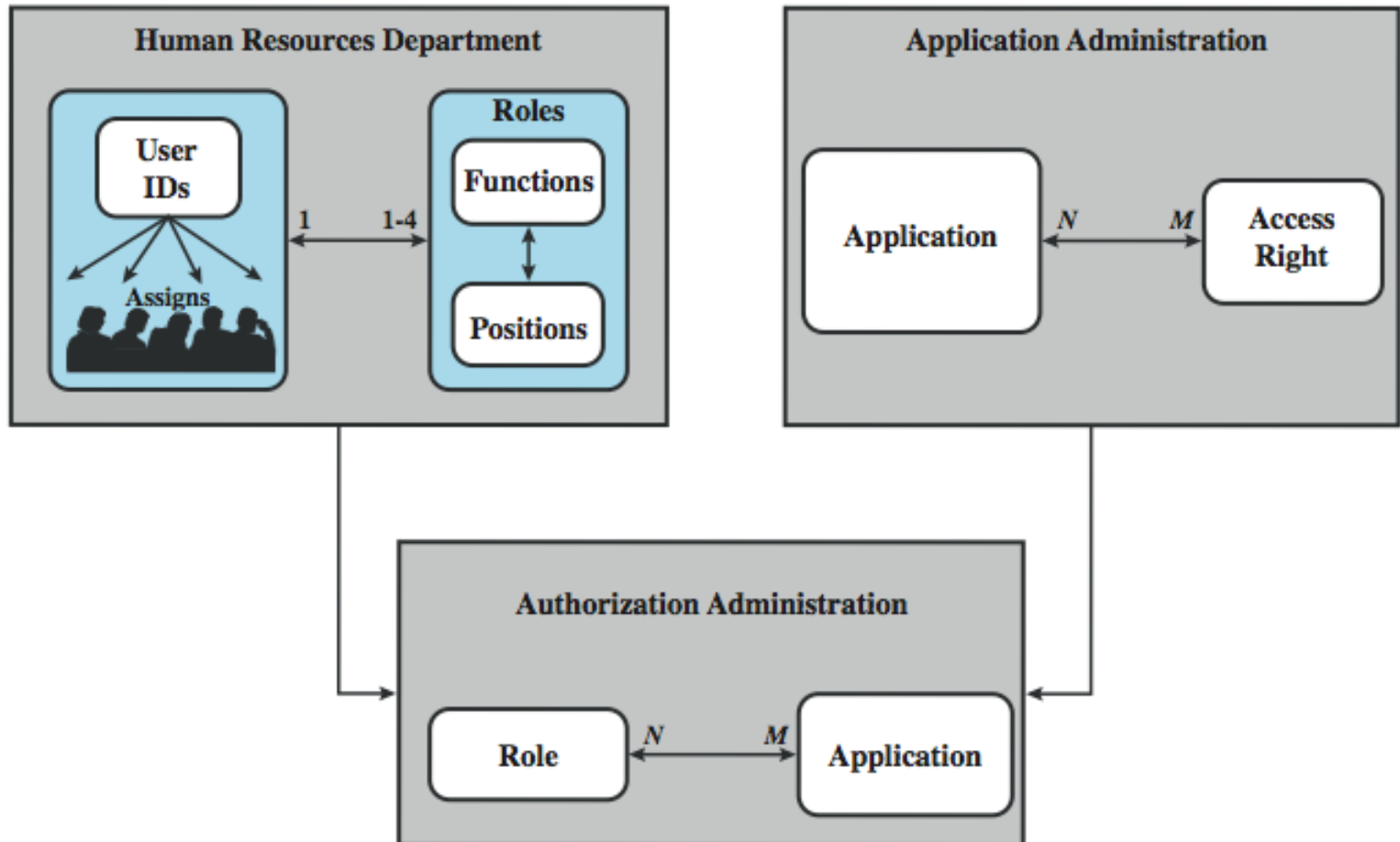
(b) RBAC models

NIST RBAC Model



SSD = static separation of duty
DSD = dynamic separation of duty

RBAC For a Bank



Security Policy

- Policy partitions system states into:
 - Authorized (secure)
 - These are states the system can enter
 - Unauthorized (nonsecure)
 - If the system enters any of these states, it's a security violation
- Secure system
 - Starts in authorized state
 - Never enters unauthorized state

Confidentiality

- X set of entities, I information
- I has *confidentiality* property with respect to X if no $x \in X$ can obtain information from I
- I can be disclosed to others
- Example:
 - X set of students
 - I final exam answer key
 - I is confidential with respect to X if students cannot obtain final exam answer key

Integrity

- X set of entities, I information
- I has *integrity* property with respect to X if all $x \in X$ trust information in I
- Types of integrity:
 - trust I , its conveyance and protection (data integrity)
 - I information about origin of something or an identity (origin integrity, authentication)
 - I resource: means resource functions as it should (assurance)

Availability

- X set of entities, I resource
- I has *availability* property with respect to X if all $x \in X$ can access I
- Types of availability:
 - traditional: x gets access or not
 - quality of service: promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved

Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
 - Security levels in multilevel security models
 - Separation of duty in Clark-Wilson model
 - Conflict of interest in Chinese Wall model

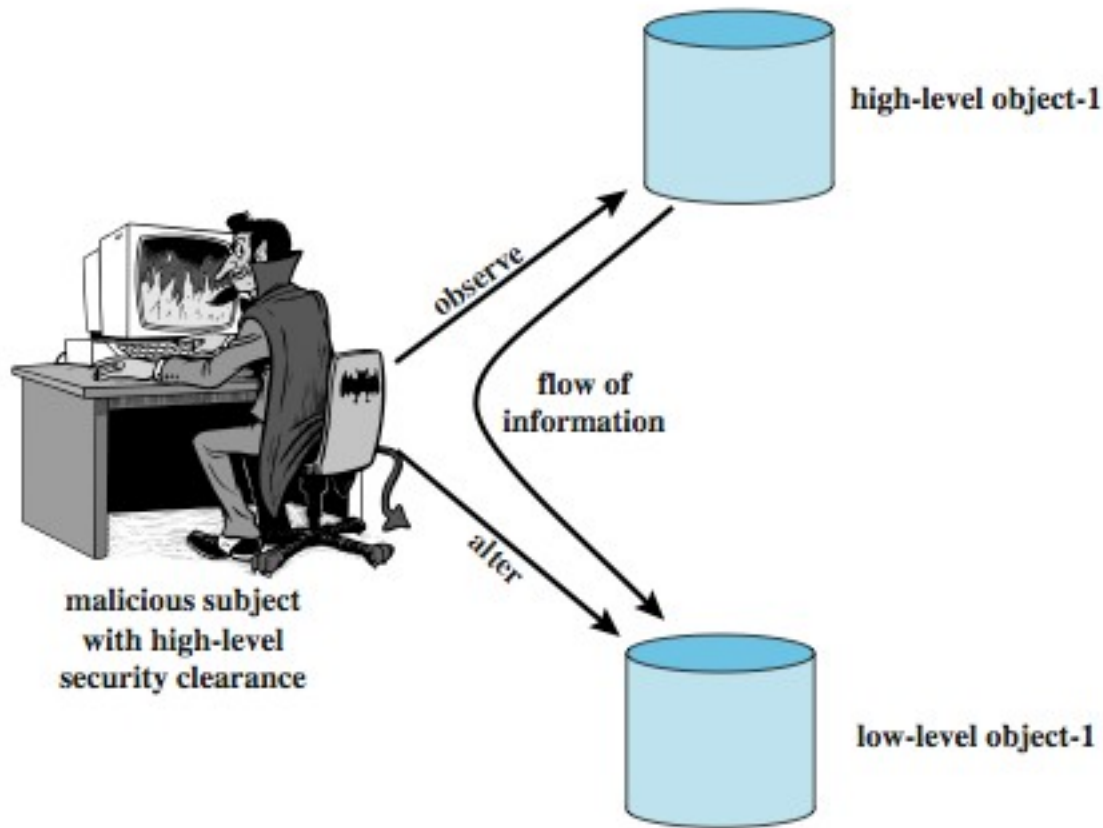
Types of Security Policies

- Military (governmental) security policy
 - Policy primarily protecting confidentiality
- Commercial security policy
 - Policy primarily protecting integrity
- Confidentiality policy
 - Policy protecting only confidentiality
- Integrity policy
 - Policy protecting only integrity

Integrity and Transactions

- Begin in consistent state
 - “Consistent” defined by specification
- Perform series of actions (*transaction*)
 - Actions cannot be interrupted
 - If actions complete, system in consistent state
 - If actions do not complete, system reverts to beginning (consistent) state

Multi-Level Security



Confidentiality Policies

- Overview
 - What is a confidentiality model
- Bell-LaPadula Model
 - General idea
 - Informal description of rules

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
 - Deals with information flow
 - Integrity incidental
- Multi-level security models are best-known examples
 - Bell-LaPadula Model basis for many, or most, of these

Bell-LaPadula Model, Step 1

- Security levels arranged in linear ordering
 - Top Secret: highest
 - Secret
 - Confidential
 - Unclassified: lowest
- Levels consist of *security clearance* $L(s)$
 - Objects have *security classification* $L(o)$

Example

<i>security level</i>	<i>subject</i>	<i>object</i>
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Ursula	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Ursula can only read Telephone Lists

Reading Information

- Information flows *up*, not *down*
 - “Reads up” disallowed, “reads down” allowed
- Simple Security Condition (Step 1)
 - Subject s can read object o iff $L(o) \leq L(s)$ and s has permission to read o
 - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
 - Sometimes called “no reads up” rule

Writing Information

- Information flows up, not down
 - “Writes up” allowed, “writes down” disallowed
- *-Property (Step 1)
 - Subject s can write object o iff $L(s) \leq L(o)$ and s has permission to write o
 - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
 - Sometimes called “no writes down” rule

Basic Security Theorem, Step 1

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition, step 1, and the *-property, step 1, then every state of the system is secure
 - Proof: induct on the number of transitions

Bell-LaPadula Model, Step 2

- Expand notion of security level to include categories
- Security level is (*clearance*, *category set*)
- Examples
 - (Top Secret, { NUC, EUR, US })
 - (Confidential, { EUR, US })
 - (Secret, { NUC, US })

Levels and Lattices

- $(A, C) \text{ dom } (A', C')$ iff $A' \leq A$ and $C' \subseteq C$
- Examples
 - $(\text{Top Secret}, \{\text{NUC}, \text{US}\}) \text{ dom } (\text{Secret}, \{\text{NUC}\})$
 - $(\text{Secret}, \{\text{NUC}, \text{EUR}\}) \text{ dom } (\text{Confidential}, \{\text{NUC}, \text{EUR}\})$
 - $(\text{Top Secret}, \{\text{NUC}\}) \not\text{dom } (\text{Confidential}, \{\text{EUR}\})$
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, dom form lattice
 - $\text{lub}(L) = (\max(A), C)$
 - $\text{glb}(L) = (\min(A), \emptyset)$

Levels and Ordering

- Security levels partially ordered
 - Any pair of security levels may (or may not) be related by *dom*
- “dominates” serves the role of “greater than” in step 1
 - “greater than” is a total ordering, though

Reading Information

- Information flows *up*, not *down*
 - “Reads up” disallowed, “reads down” allowed
- Simple Security Condition (Step 2)
 - Subject s can read object o iff $L(s) \text{ dom } L(o)$ and s has permission to read o
 - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
 - Sometimes called “no reads up” rule

Writing Information

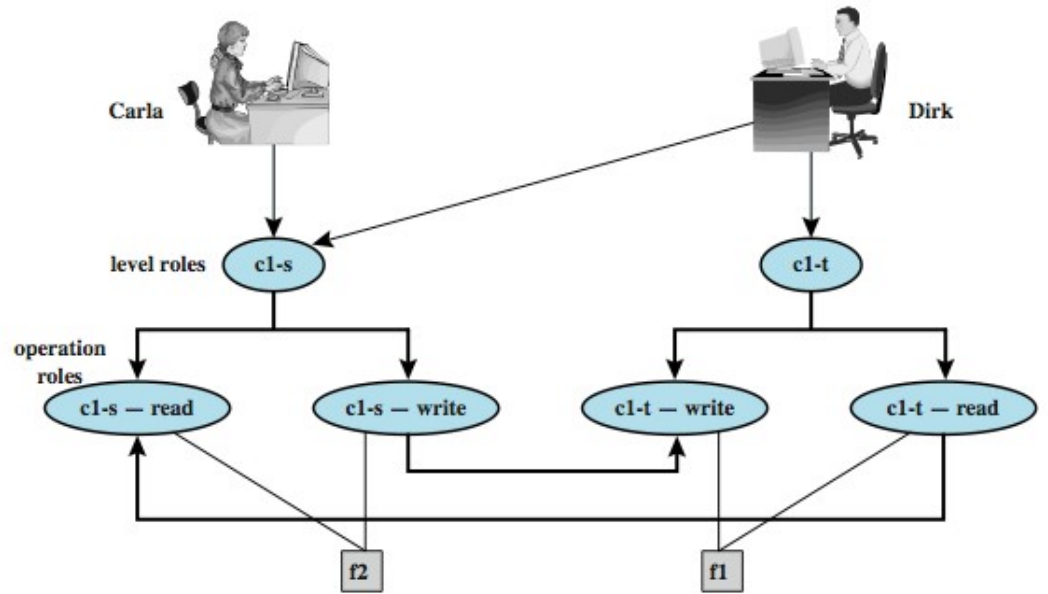
- Information flows up, not down
 - “Writes up” allowed, “writes down” disallowed
- *-Property (Step 2)
 - Subject s can write object o iff $L(o) \text{ dom } L(s)$ and s has permission to write o
 - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
 - Sometimes called “no writes down” rule

Basic Security Theorem, Step 2

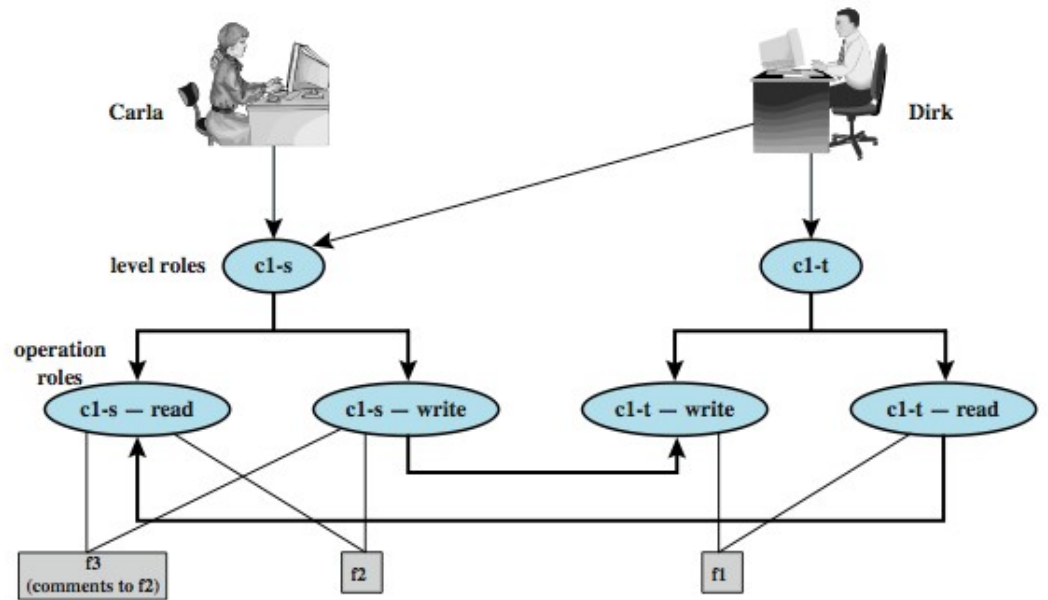
- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition, step 2, and the *-property, step 2, then every state of the system is secure
 - Proof: induct on the number of transitions
 - In actual Basic Security Theorem, discretionary access control treated as third property, and simple security property and *-property phrased to eliminate discretionary part of the definitions — but simpler to express the way done here.

BLP

Example



(a) Two new files are created: $f1$: $c1-t$; $f2$: $c1-s$

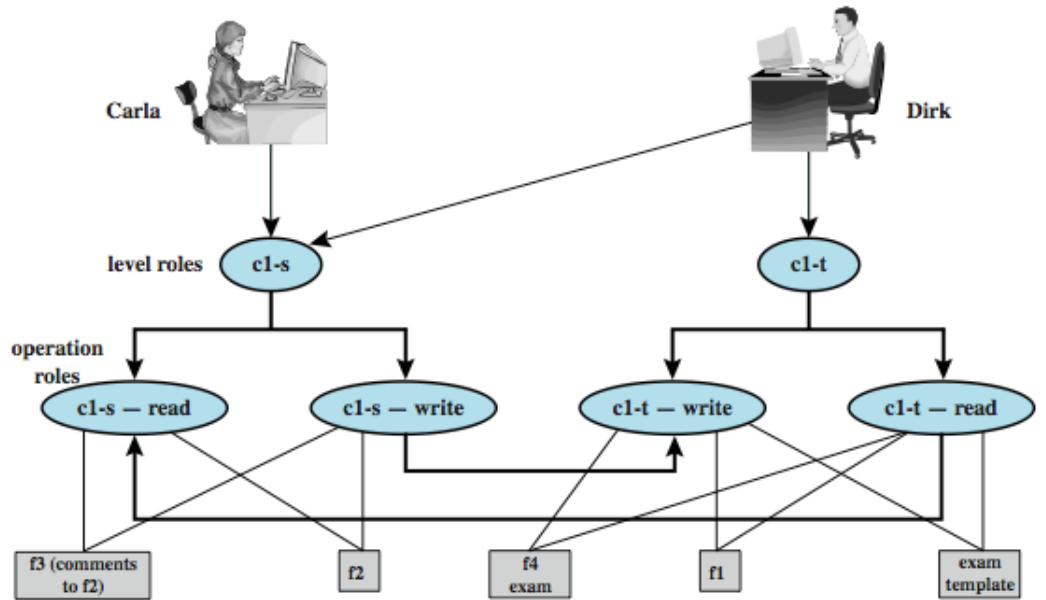


(b) A third file is added: $f3$: $c1-s$

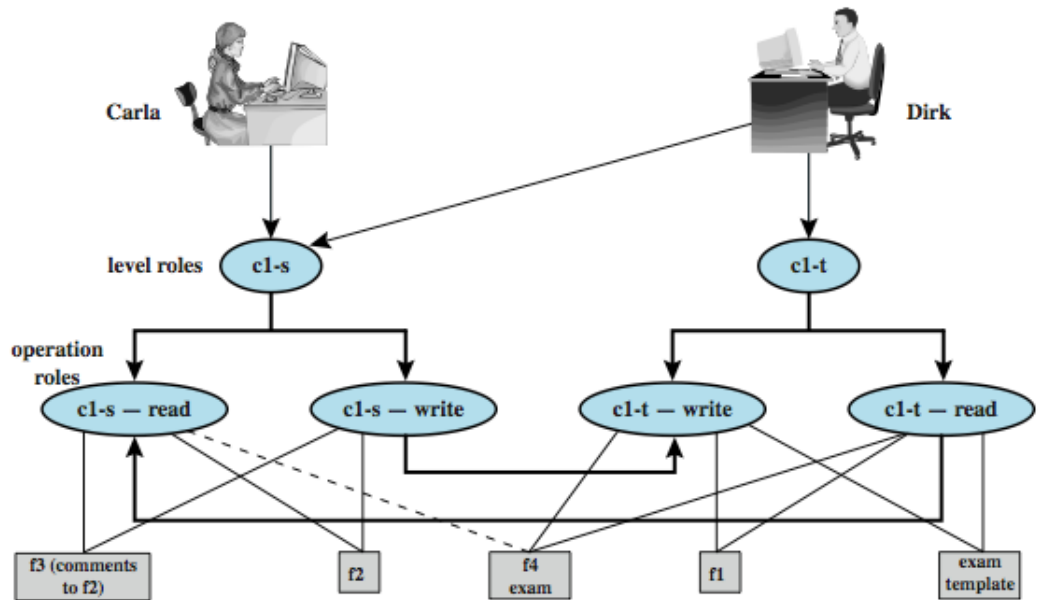
BLP

Example

cont.



(c) An exam is created based on an existing template: $f4$: $c1-t$

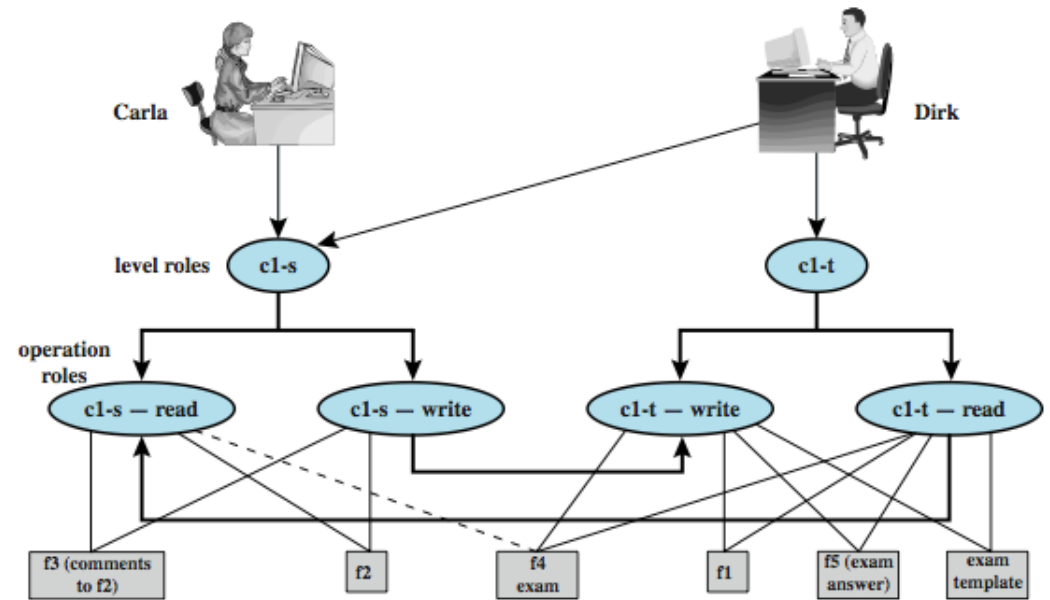


(d) Carla, as student, is permitted access to the exam: $f4$: $c1-s$

BLP

Example

cont.



(e) The answers given by Carla are only accessible for the teacher: $f5$: $c1-t$

Biba Integrity Model

- Set of subjects S , objects O , integrity levels I , relation $\leq \subseteq I \times I$ holding when second dominates first
- $min: I \times I \rightarrow I$ returns lesser of integrity levels
- $i: S \cup O \rightarrow I$ gives integrity level of entity
- $\underline{r}: S \times O$ means $s \in S$ can read $o \in O$
- \underline{w} , \underline{x} defined similarly

Intuition for Integrity Levels

- The higher the level, the more confidence
 - That a program will execute correctly
 - That data is accurate and/or reliable
- Note relationship between integrity and trustworthiness
- Important point: *integrity levels are **not** security levels*

Integrity policies: Biba's Model

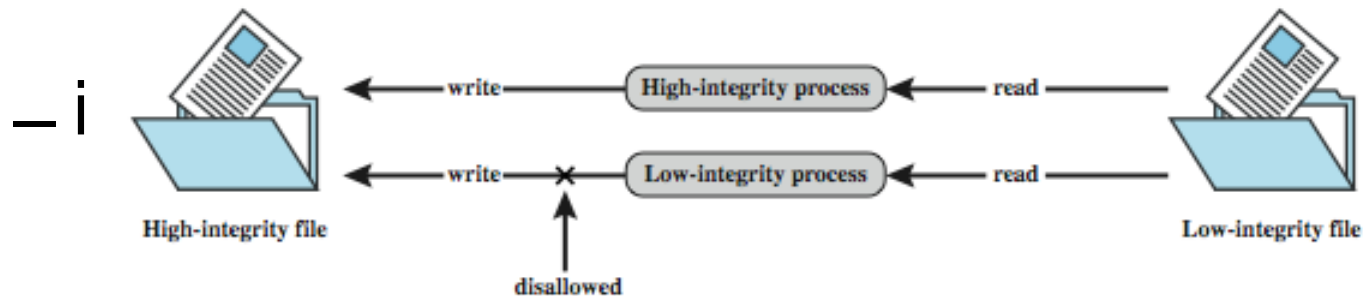
- Similar to Bell-LaPadula model
 1. $s \in S$ can read $o \in O$ iff $i(s) \leq i(o)$
 2. $s \in S$ can write to $o \in O$ iff $i(o) \leq i(s)$
 3. $s_1 \in S$ can execute $s_2 \in S$ iff $i(s_2) \leq i(s_1)$
- Information flow result holds
 - Different proof, though

Biba Integrity Model

- strict integrity policy:

– simple integrity: $I(S) \geq I(O)$

– integrity confinement: $I(S) \leq I(O)$



Clark-Wilson Integrity Model

- Integrity defined by a set of constraints
 - Data in a *consistent* or *valid* state when it satisfies these
- Example: Bank
 - D today's deposits, W withdrawals, YB yesterday's balance, TB today's balance
 - Integrity constraint: $D + YB - W$
- *Well-formed transaction* move system from one consistent state to another
- Issue: who examines, certifies transactions done correctly?

Entities

- CDIs: constrained data items
 - Data subject to integrity controls
- UDIs: unconstrained data items
 - Data not subject to integrity controls
- IVPs: integrity verification procedures
 - Procedures that test the CDIs conform to the integrity constraints
- TPs: transaction procedures
 - Procedures that take the system from one valid state to another

Certification Rules 1 and 2

- CR1 When any IVP is run, it must ensure all CDIs are in a valid state
- CR2 For some associated set of CDIs, a TP must transform those CDIs in a valid state into a (possibly different) valid state
- Defines relation *certified* that associates a set of CDIs with a particular TP
 - Example: TP balance, CDIs accounts, in bank example

Enforcement Rules 1 and 2

- ER1 The system must maintain the certified relations and must ensure that only TPs certified to run on a CDI manipulate that CDI.
- ER2 The system must associate a user with each TP and set of CDIs. The TP may access those CDIs on behalf of the associated user. The TP cannot access that CDI on behalf of a user not associated with that TP and CDI.
- System must maintain, enforce certified relation
 - System must also restrict access based on user ID (*allowed* relation)

Users and Rules

- CR3 The allowed relations must meet the requirements imposed by the principle of separation of duty.
- ER3 The system must authenticate each user attempting to execute a TP
- Type of authentication undefined, and depends on the instantiation
 - Authentication *not* required before use of the system, but *is* required before manipulation of CDIs (requires using TPs)

Logging

CR4 All TPs must append enough information to reconstruct the operation to an append-only CDI.

- This CDI is the log
- Auditor needs to be able to determine what happened during reviews of transactions

Handling Untrusted Input

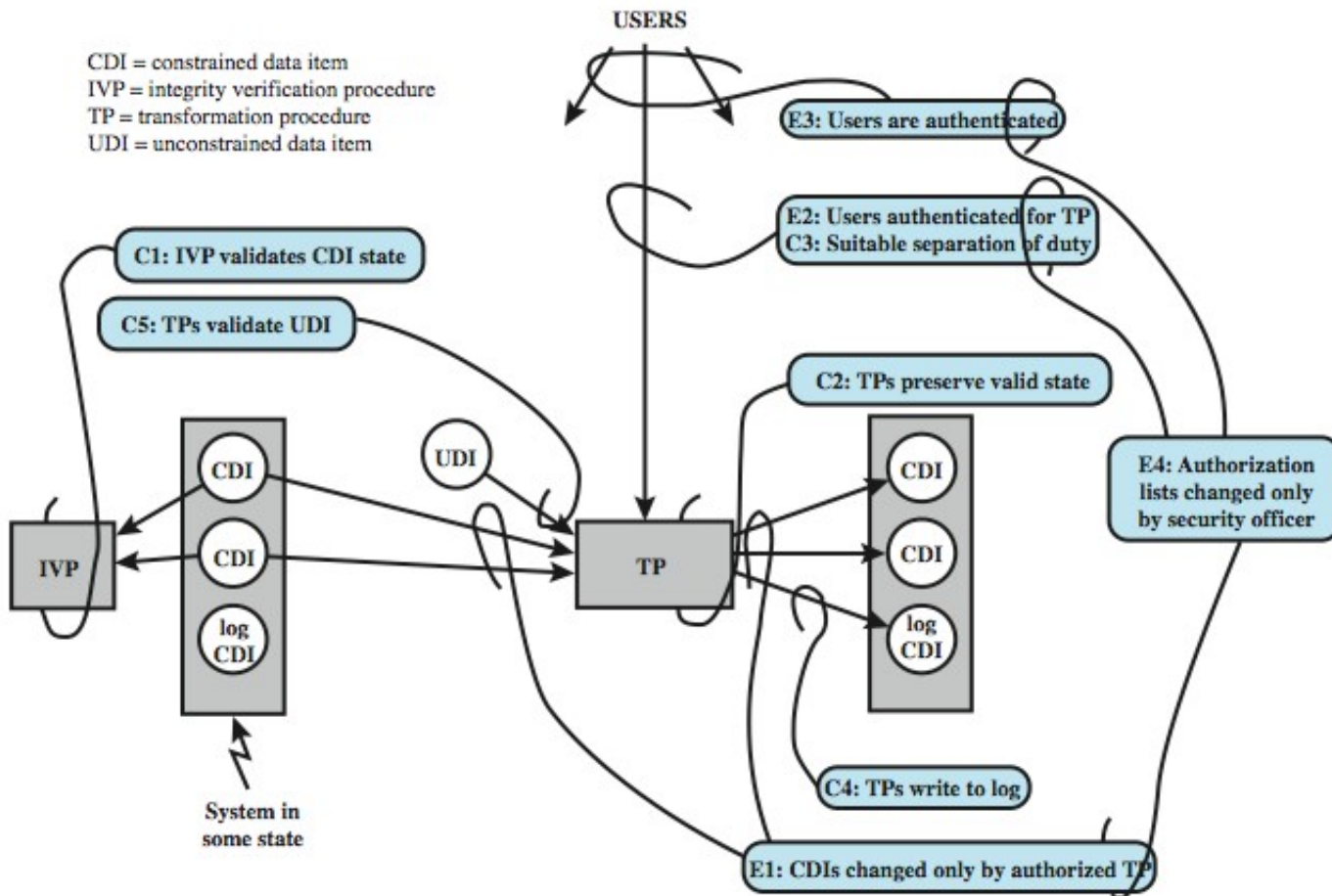
- CR5 Any TP that takes as input a UDI may perform only valid transformations, or no transformations, for all possible values of the UDI. The transformation either rejects the UDI or transforms it into a CDI.
- In bank, numbers entered at keyboard are UDIs, so cannot be input to TPs. TPs must validate numbers (to make them a CDI) before using them; if validation fails, TP rejects UDI

Separation of Duty In Model

ER4 Only the certifier of a TP may change the list of entities associated with that TP. No certifier of a TP, or of an entity associated with that TP, may ever have execute permission with respect to that entity.

- Enforces separation of duty with respect to certified and allowed relations

Clark-Wilson Integrity Model



Comparison With Requirements

1. Users can't certify TPs, so CR5 and ER4 enforce this
2. Procedural, so model doesn't directly cover it; but special process corresponds to using TP
 - No technical controls can prevent programmer from developing program on production system; usual control is to delete software tools
3. TP does the installation, trusted personnel do certification

Comparison With Requirements

4. CR4 provides logging; ER3 authenticates trusted personnel doing installation; CR5, ER4 control installation procedure
 - New program UDI before certification, CDI (and TP) after
2. Log is CDI, so appropriate TP can provide managers, auditors access
 - Access to state handled similarly

Comparison to Biba

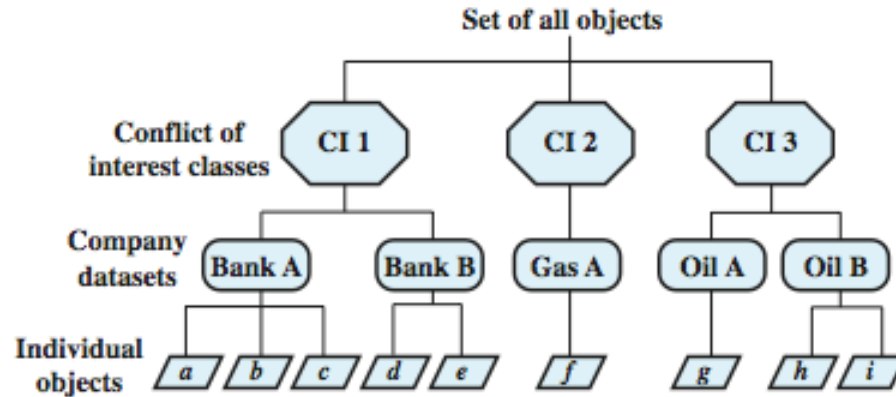
- Biba
 - No notion of certification rules; trusted subjects ensure actions obey rules
 - Untrusted data examined before being made trusted
- Clark-Wilson
 - Explicit requirements that *actions* must meet
 - Trusted entity must certify *method* to upgrade untrusted data (and not certify the data itself)

Chinese Wall Model

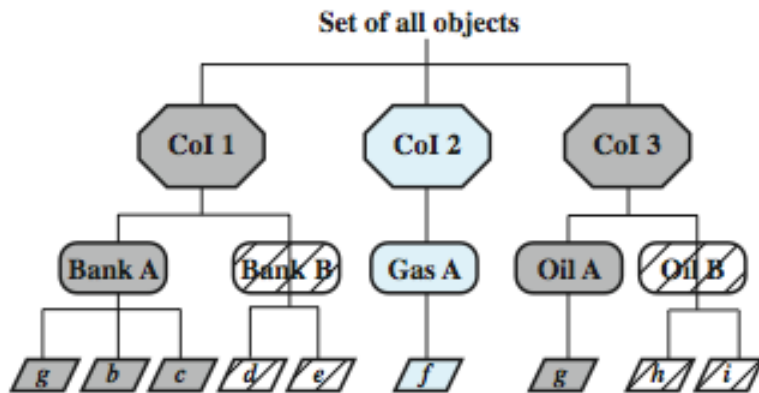
Problem:

- Tony advises American Bank about investments
- He is asked to advise Toyland Bank about investments
- Conflict of interest to accept, because his advice for either bank would affect his advice to the other bank

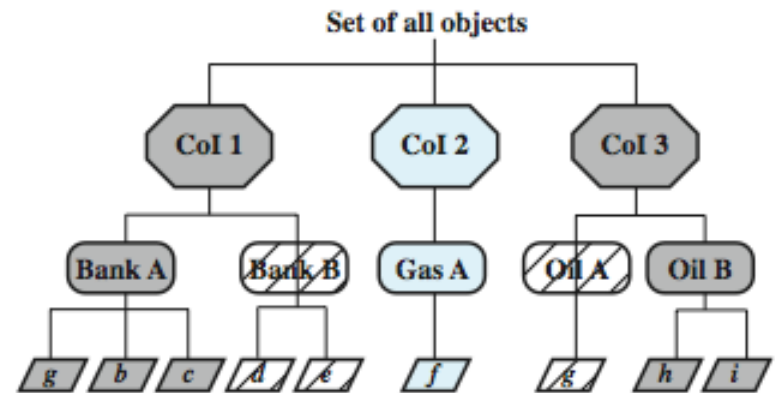
Chinese Wall Model



(a) Example set



(b) John has access to Bank A and Oil A



(c) Jane has access to Bank A and Oil B

Organization

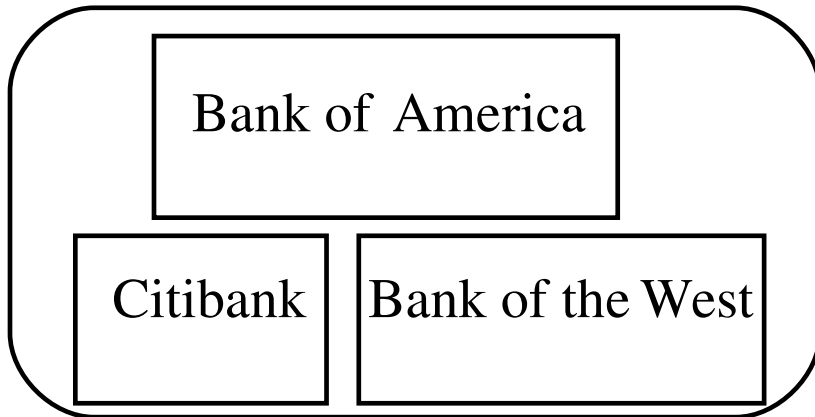
- Organize entities into “conflict of interest” classes
- Control subject accesses to each class
- Control writing to all classes to ensure information is not passed along in violation of rules
- Allow sanitized data to be viewed by everyone

Definitions

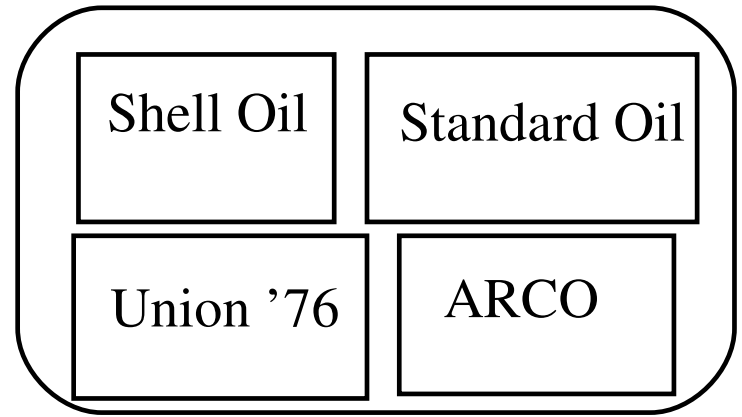
- *Objects*: items of information related to a company
- *Company dataset* (CD): contains objects related to a single company
 - Written $CD(O)$
- *Conflict of interest class* (COI): contains datasets of companies in competition
 - Written $COI(O)$
 - Assume: each object belongs to exactly one *COI* class

Example

Bank COI Class



Gasoline Company COI Class



Temporal Element

- If Anthony reads any CD in a COI, he can *never* read another CD in that COI
 - Possible that information learned earlier may allow him to make decisions later
 - Let $PR(S)$ be set of objects that S has already read

CW-Simple Security Condition

- s can read o iff either condition holds:
 1. There is an o' such that s has accessed o' and $CD(o') = CD(o)$
 - Meaning s has read something in o 's dataset
 2. For all $o' \in O$, $o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$
 - Meaning s has not read any objects in o 's conflict of interest class
- Ignores sanitized data (see below)
- Initially, $PR(s) = \emptyset$, so initial read request granted

Sanitization

- Public information may belong to a CD
 - As is publicly available, no conflicts of interest arise
 - So, should not affect ability of analysts to read
 - Typically, all sensitive data removed from such information before it is released publicly (called *sanitization*)
- Add third condition to CW-Simple Security Condition:
 3. o is a sanitized object

Writing

- Anthony, Susan work in same trading house
- Anthony can read Bank 1's CD, Gas' CD
- Susan can read Bank 2's CD, Gas' CD
- If Anthony could write to Gas' CD, Susan can read it
 - Hence, indirectly, she can read information from Bank 1's CD, a clear conflict of interest

CW-^{*}-Property

- s can write to o iff both of the following hold:
 1. The CW-simple security condition permits s to read o ; and
 2. For all *unsanitized* objects o' , if s can read o' , then $CD(o') = CD(o)$
- Says that s can write to an object if all the (unsanitized) objects it can read are in the same dataset

Compare to Bell-LaPadula

- Fundamentally different
 - CW has no security labels, B-LP does
 - CW has notion of past accesses, B-LP does not
- Bell-LaPadula can capture state at any time
 - Each (COI, CD) pair gets security category
 - Two clearances, S (sanitized) and U (unsanitized)
 - $S \text{ dom } U$
 - Subjects assigned clearance for compartments without multiple categories corresponding to CDs in same COI class

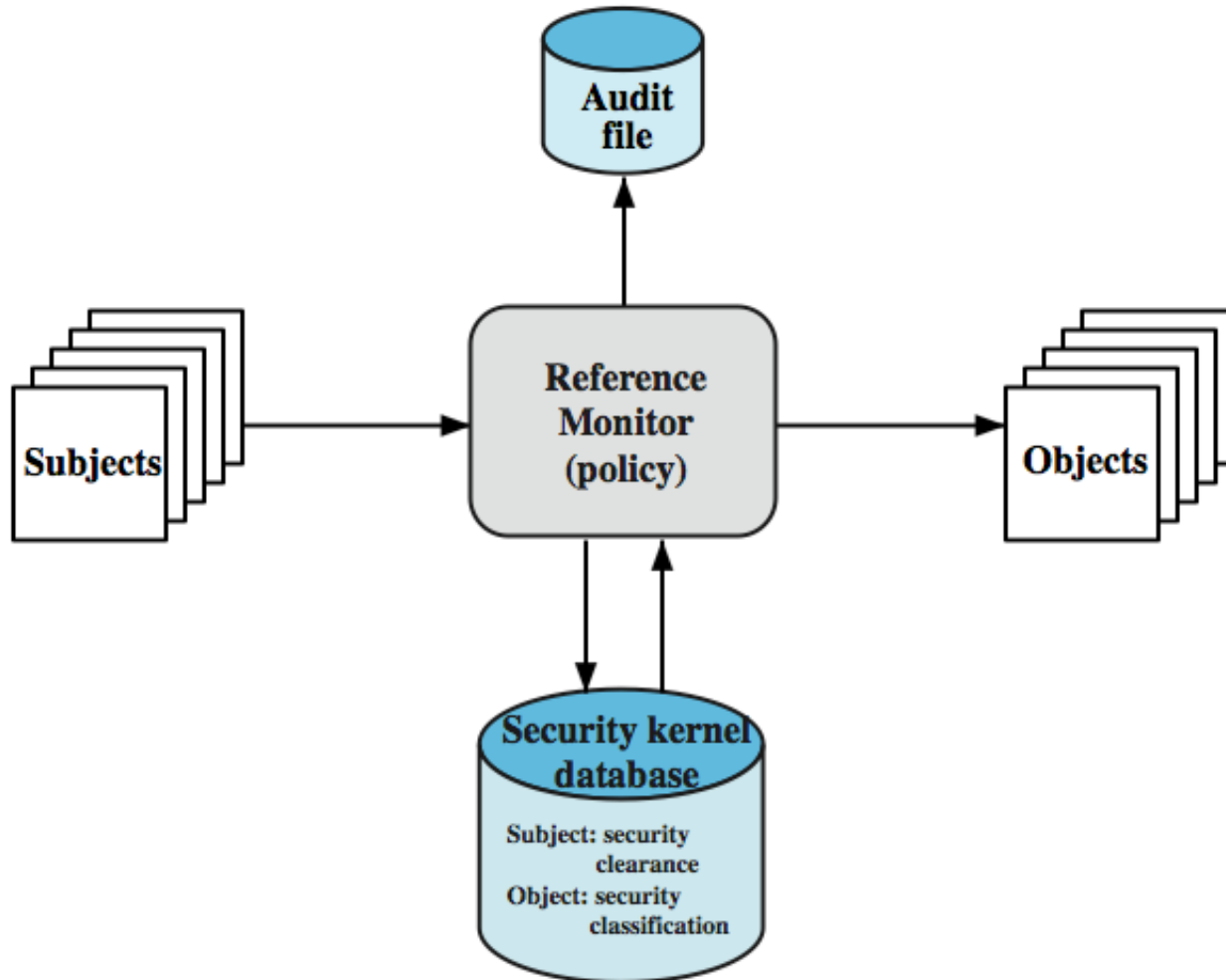
Compare to Bell-LaPadula

- Bell-LaPadula cannot track changes over time
 - Susan becomes ill, Anna needs to take over
 - C-W history lets Anna know if she can
 - No way for Bell-LaPadula to capture this
- Access constraints change over time
 - Initially, subjects in C-W can read any object
 - Bell-LaPadula constrains set of objects that a subject can access
 - Can't clear all subjects for all categories, because this violates CW-simple security condition

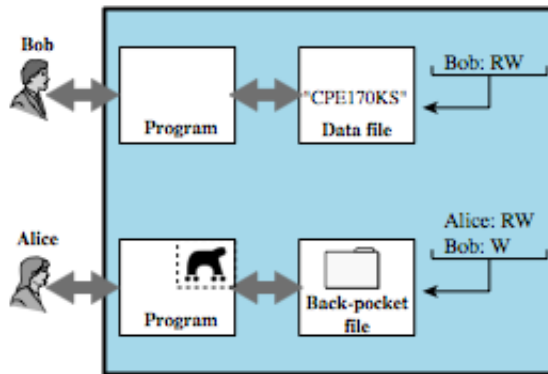
Compare to Clark-Wilson

- Clark-Wilson Model covers integrity, so consider only access control aspects
- If “subjects” and “processes” are interchangeable, a single person could use multiple processes to violate CW-simple security condition
 - Would still comply with Clark-Wilson Model
- If “subject” is a specific person and includes all processes the subject executes, then consistent with Clark-Wilson Model

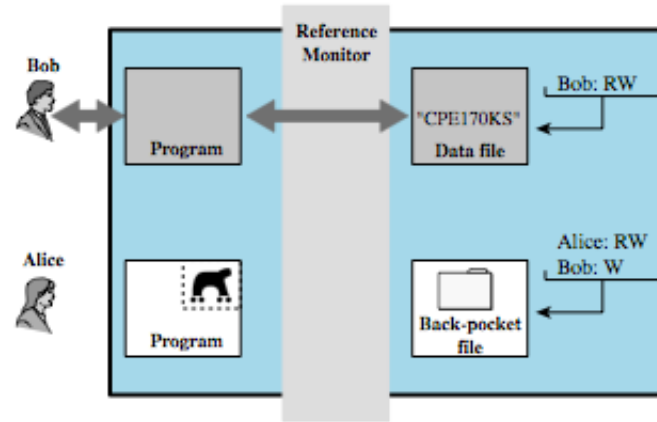
Reference Monitors



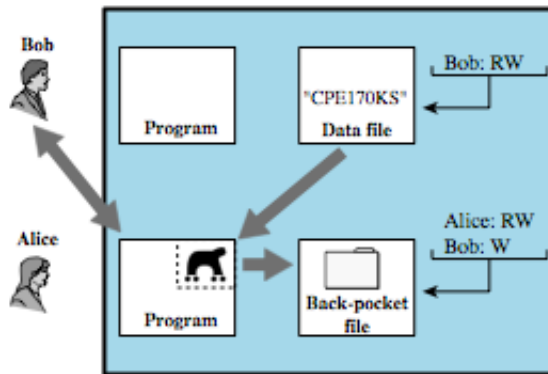
Trojan Horse Defence



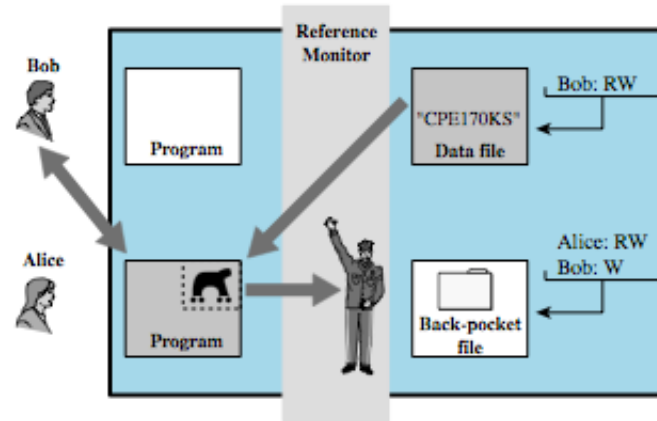
(a)



(c)



(b)



(d)

Multilevel Security (MLS)

- a class of system that has system resources (particularly stored information) at more than one security level (i.e., has different types of sensitive resources) and that permits concurrent access by users who differ in security clearance and need-to-know, but is able to prevent each user from accessing resources for which the user lacks authorization.

MLS Security for Role-Based Access Control

- rule based access control (RBAC) can implement BLP MLS rules given:
 - security constraints on users
 - constraints on read/write permissions
 - read and write level role access definitions

Summary

- Bell-Lapadula security model
- other models
- reference monitors & trojan horse defence
- multilevel secure RBAC and databases