Integrity Policies

CS461/ECE422 – Computer Security I

Fall 2009

Reading

- CS: Chapter 6

Overview

- Requirements
  - Very different than confidentiality policies
- Biba's models
  - Low-Water-Mark policy
  - Ring policy
  - Strict Integrity policy
- Lipner's model
  - Combines Bell-LaPadula, Biba
- Clark-Wilson model

Requirements of Integrity Policies

1. Users will not write their own programs, but will use existing production programs and databases.
2. Programmers will develop and test programs on a non-production system; if they need access to actual data, they will be given production data via a special process, but will use it on their development system.
3. A special process must be followed to install a program from the development system onto the production system.
4. The special process in requirement 3 must be controlled and audited.
5. The managers and auditors must have access to both the system state and the system logs that are generated.

Lipner 82

Biba Integrity Model

Basis for all 3 models:

- 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
- $\underline{r} \subseteq S \times O$ means $s \in S$ can read $o \in O$
- $\underline{w}$, $\underline{x}$ defined similarly

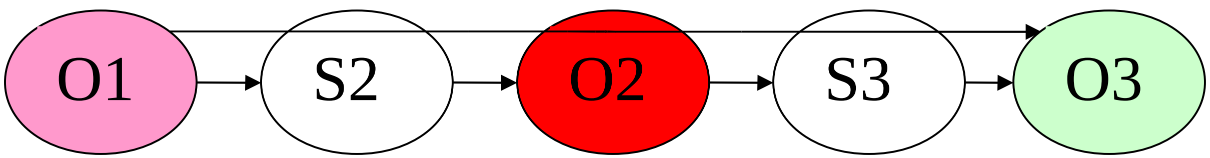
Biba 77

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*

Information Transfer Path

- An *information transfer path* is a sequence of objects $o_1, \ldots, o_{n+1}$ and corresponding sequence of subjects $s_1, \ldots, s_n$ such that s_i r o_i and s_i w o_{i+1} for all i, $1 \le i \le n$.
- Idea: information can flow from o_1 to o_{n+1} along this path by successive reads and writes

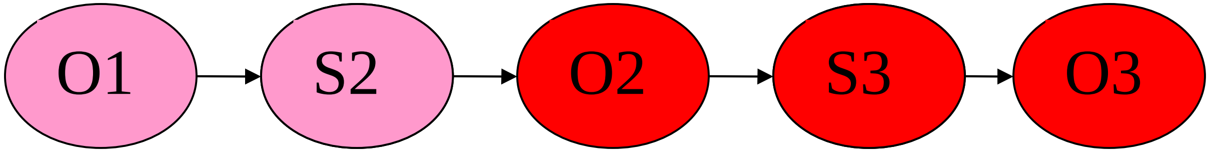


Low-Water-Mark Policy

- Idea: when s reads o, $i(s) = min(i(s), i(o))$; s can only write objects at lower levels
- Rules
  - $s \in S$ can write to $o \in O$ if and only if $i(o) \le i(s)$.
  - If $s \in S$ reads $o \in O$, then $i'(s) = min(i(s), i(o))$, where $i'(s)$ is the subject's integrity level after the read.
  - $s_1 \in S$ can execute $s_2 \in S$ if and only if $i(s_2) \le i(s_1)$.

Information Flow and Model

- If there is information transfer path from $o_1 \in O$ to $o_{n+1} \in O$, enforcement of low-water-mark policy requires $i(o_{n+1}) \leq i(o_1)$ for all n



Problems

- Subjects' integrity levels decrease as system runs
  - Soon no subject will be able to access objects at high integrity levels
- Alternative: change object levels rather than subject levels
  - Soon all objects will be at the lowest integrity level
- Crux of problem is model prevents indirect modification
  - Because subject levels lowered when subject reads from low-integrity object

Ring Policy

- Idea: subject integrity levels static
- Rules
  - $s \in S$ can write to $o \in O$ if and only if $i(o) \le i(s)$.
  - Any subject can read any object.
  - $s_1 \in S$ can execute $s_2 \in S$ if and only if $i(s_2) \le i(s_1)$.
- Eliminates indirect modification problem
- Same information flow result holds

Strict Integrity Policy

- Dual of Bell-LaPadula model
  - $s \in S$ can read $o \in O$ iff $i(s) \le i(o)$
  - $s \in S$ can write to $o \in O$ iff $i(o) \le i(s)$
  - $s_1 \in S$ can execute $s_2 \in O$ iff $i(s_2) \le i(s_1)$
- Add compartments and discretionary controls to get full dual of Bell-LaPadula model
- Information flow result holds
  - Different proof, though
- Term “Biba Model” refers to this

Execute Clarification

- What is the label of the new process created as result of executing a file?
  - In a real implementation would probably have mechanisms for choosing label of invoking process, label of executable, or some combination.
    - see Trusted OS slides
  - Labeling new files has similar points of confusion
- For the base case, assume new process inherit integrity label of invoking process
  - This would be the minimum of the two labels

LOCUS and Biba

- Goal: prevent untrusted software from altering data or other software
- Approach: make levels of trust explicit
  - *credibility rating* based on estimate of software's trustworthiness (0 untrusted, *n* highly trusted)
  - *trusted file systems* contain software with a single credibility level
  - Process has *risk level* or highest credibility level at which process can execute
  - Must use *run-untrusted* command to run software at lower credibility level

Pozzo, Gray 86

Integrity Matrix Model

- Lipner proposed this as first realistic commercial model
- Combines Bell-LaPadula, Biba models to obtain model conforming to requirements
- Do it in two steps
  - Bell-LaPadula component first
  - Add in Biba component

Lipner 82

Bell-LaPadula Clearances

- 2 security clearances/classifications
  - AM (Audit Manager): system audit, management functions
  - SL (System Low): any process can read at this level

Bell-LaPadula Categories

- 5 categories
  - D (Development): production programs in development but not yet in use
  - PC (Production Code): production processes, programs
  - PD (Production Data): data covered by integrity policy
  - SD (System Development): system programs in development but not yet in use
  - T (Software Tools): programs on production system not related to protected data

Users and Security Levels

Subjects	**Security Level**
Ordinary users	(SL, { PC, PD })
Application developers	(SL, { D, T })
System programmers	(SL, { SD, T })
System managers and auditors	(AM, { D, PC, PD, SD, T })
System controllers	(SL, {D, PC, PD, SD, T}) and downgrade privilege

Objects and Classifications

Objects	**Security Level**
Development code/test data	(SL, { D, T })
Production code	(SL, { PC })
Production data	(SL, { PC, PD })
Software tools	(SL, { T })
System programs	(SL, ∅)
System programs in modification	(SL, { SD, T })
System and application logs	(AM, { *appropriate* })

Ideas

- Ordinary users can execute (read) production code but cannot alter it
- Ordinary users can alter and read production data
- System managers need access to all logs but cannot change levels of objects
- System controllers need to install code (hence downgrade capability)
- Logs are append only, so must dominate subjects writing them

Check Requirements

1. Users have no access to T, so cannot write their own programs
2. Applications programmers have no access to PD, so cannot access production data; if needed, it must be put into D, requiring the system controller to intervene
3. Installing a program requires downgrade procedure (from D to PC), so only system controllers can do it

More Requirements

4. Control: only system controllers can downgrade; audit: any such downgrading must be altered
5. System management and audit users are in AM and so have access to system state and logs

Problem

- Too inflexible
  - An application developer cannot run a program for repairing inconsistent or erroneous production database
    - Application programmers are not given access to production data
- So add more …

Adding Biba

- 3 integrity classifications
  - ISP (System Program): for system programs
  - IO (Operational): production programs, development software
  - ISL (System Low): users get this on log in
- 2 integrity categories
  - ID (Development): development entities
  - IP (Production): production entities

Simplify Bell-LaPadula

- Reduce security categories to 3:
  - SP (Production): production code, data
  - SD (Development): same as D
  - SSD (System Development): same as old SD

Users and Levels

Subjects	**Security Level**	**Integrity Level**
Ordinary users	(SL, { SP })	(ISL, { IP })
Application developers	(SL, { SD })	(ISL, { ID })
System programmers	(SL, { SSD })	(ISL, { ID })
System managers and auditors	(AM, { SP, SD, SSD })	(ISL, ∅)
System controllers	(SL, { SP, SD, SSD }) and downgrade privilege	(ISP, { IP, ID})
Repair	(SL, { SP })	(ISL, { IP })

Objects and Classifications

Objects	**Security Level**	**Integrity Level**
Development code/test data	(SL, { SD })	(ISL, { ID })
Production code	(SL, { SP })	(IO, { IP })
Production data	(SL, { SP })	(ISL, { IP })
Software tools	(SL, $\varnothing$)	(IO, { ID })
System programs	(SL, $\varnothing$)	(ISP, { IP, ID })
System programs in modification	(SL, { SSD })	(ISL, { ID })
System and application logs	(AM, { *appropriate* })	(ISL, $\varnothing$)
Repair	(SL, {SP})	(ISL, { IP })

Ideas

- Security clearances of subjects same as without integrity levels
- Ordinary users need to modify production data, so ordinary users must have write access to integrity category IP
- Ordinary users must be able to write production data but not production code; integrity classes allow this
  - Note writing constraints removed from security classes

Clark-Wilson Integrity Model

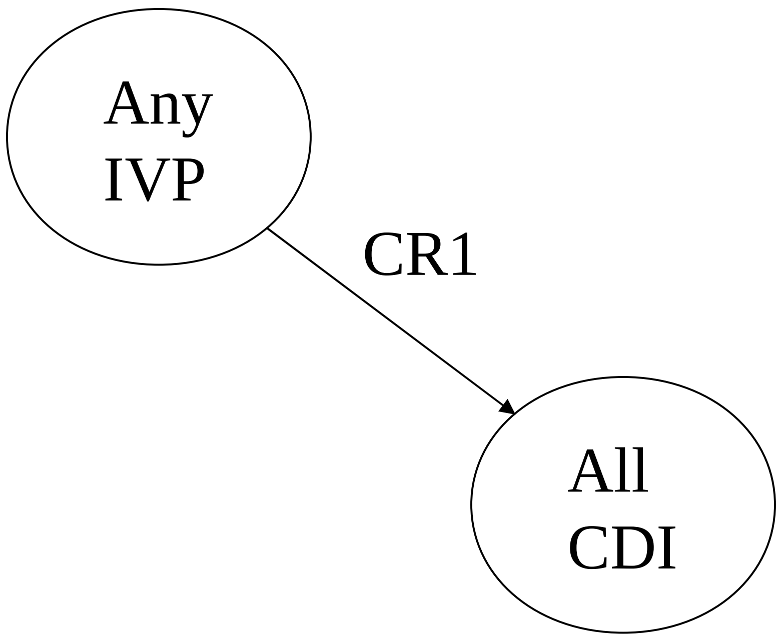
- Integrity defined by a set of constraints
  - Data in a *consistent* or valid state when it satisfies these
- Example: Bank
  - *D* today's deposits, *W* withdrawals, *YB* yesterday's balance, *TB* today's balance
  - Integrity constraint: TB = *D* + *YB* –*W*
- *Well-formed transaction* move system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

Clark, Wilson 87

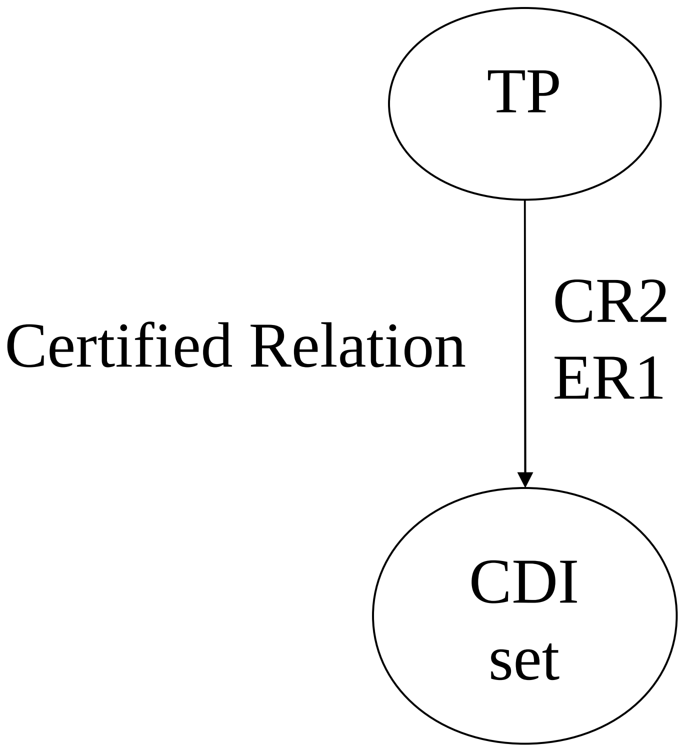
Entities

- 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

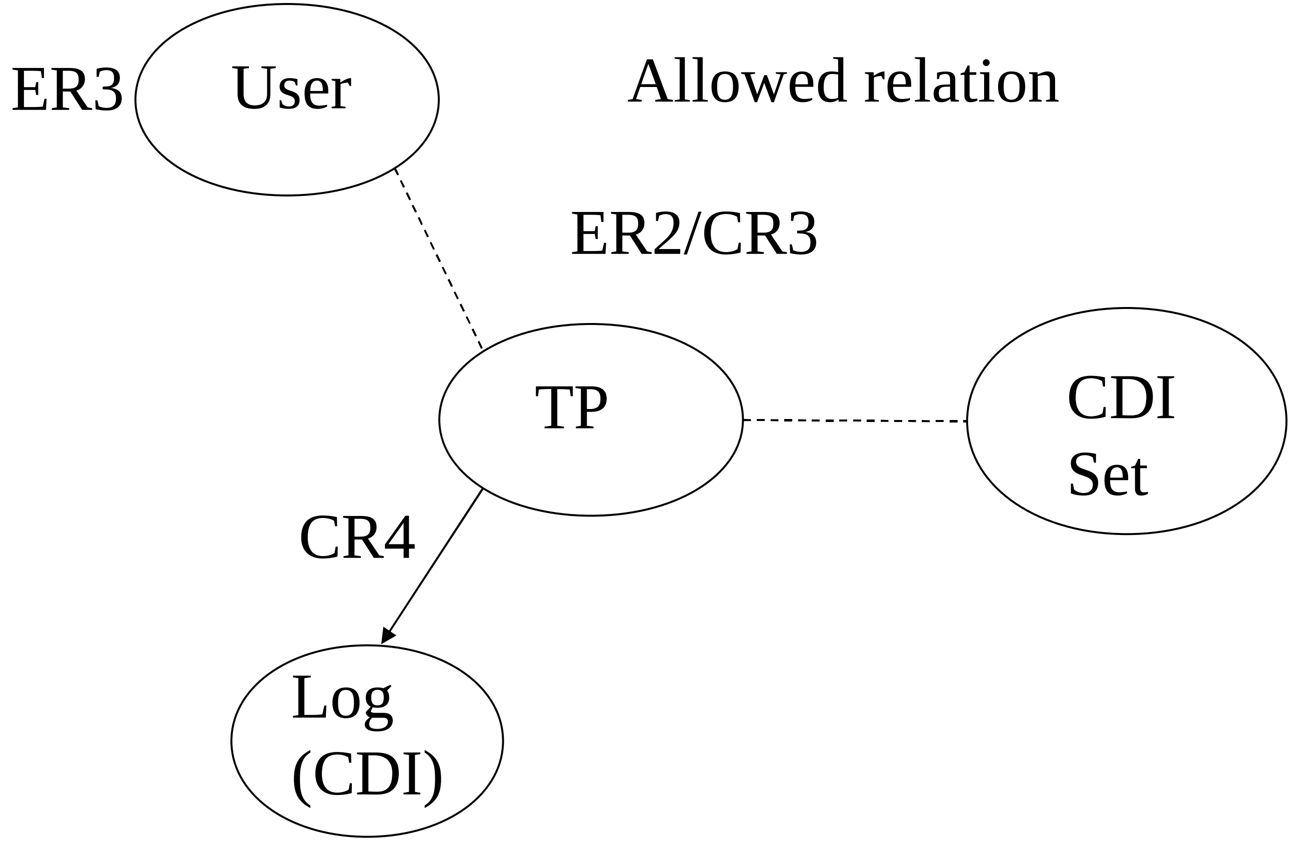
Certification Rule 1



CR2 and ER1



Other Rules



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

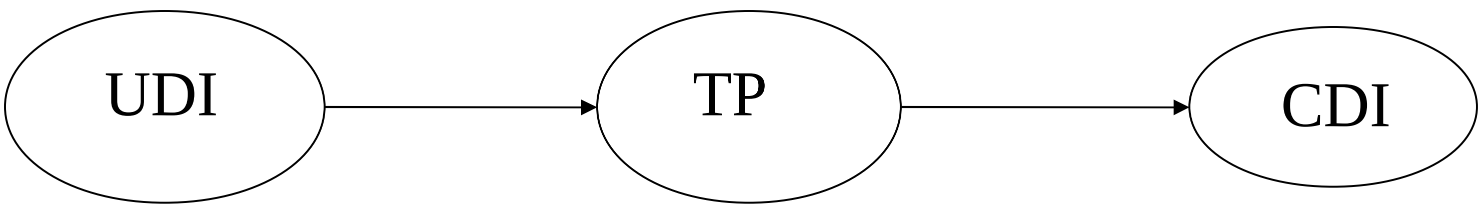
- Type of authentication undefined, and depends on the instantiation
- Authentication *not* required before use of the system, but *is* required before manipulation of CDIs (requires using TPs)

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

CR5 – Handling Untrusted Input

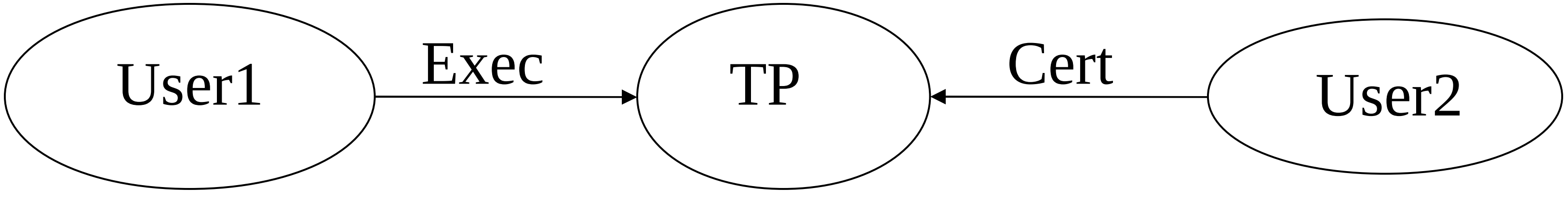


Handling Untrusted Input

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

ER4



User1 intersect User2 = empty set

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

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

1. 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)

UNIX Implementation

- Considered "allowed" relation

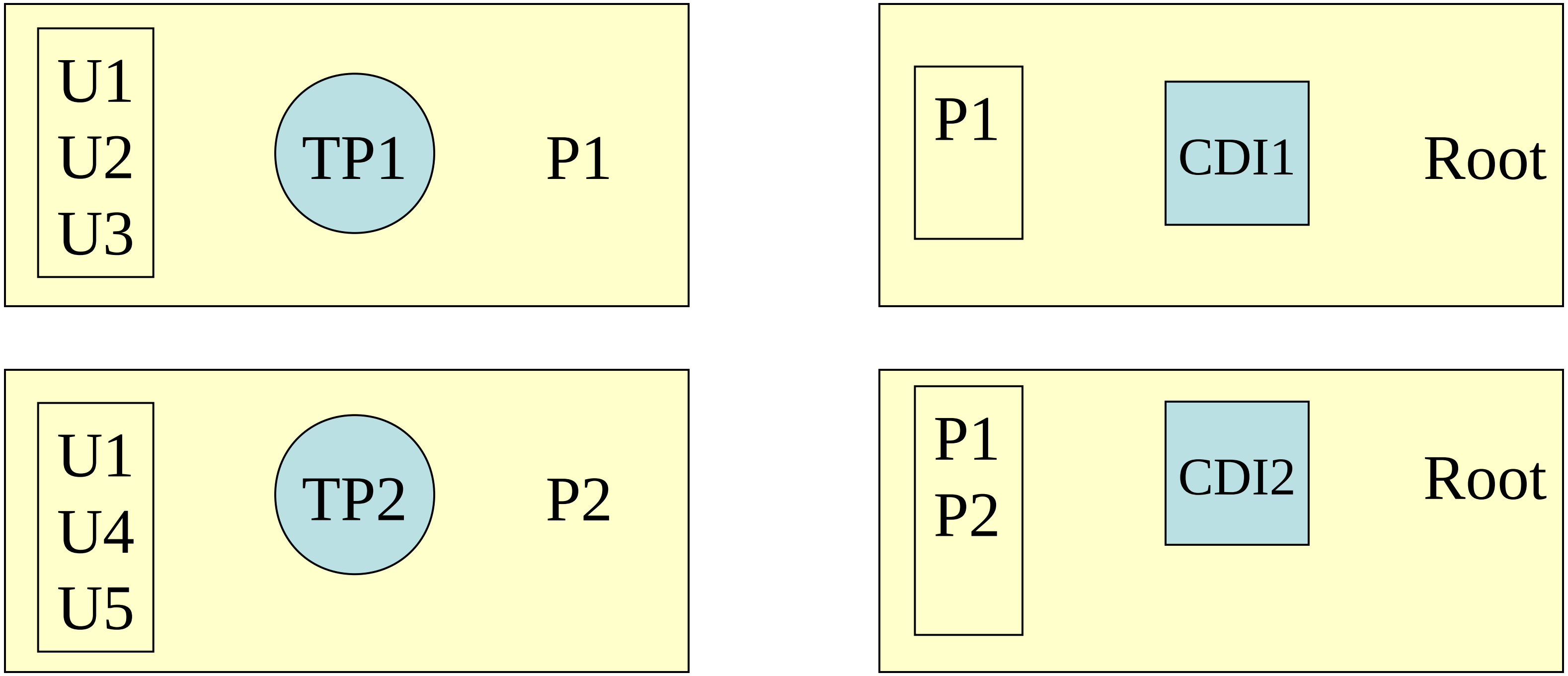
  (*user*, *TP*, { *CDI set* })
- Each TP is owned by a different user
  - These "users" are actually locked accounts, so no real users can log into them; but this provides each TP a unique UID for controlling access rights
  - TP is setuid to that user
- Each TP's group contains set of users authorized to execute TP
- Each TP is executable by group, not by world

Polk 93

CDI Arrangement

- CDIs owned by *root* or some other unique user
  - Again, no logins to that user’s account allowed
- CDI’s group contains users of TPs allowed to manipulate CDI
- Now each TP can manipulate CDIs for single user

Basic Example



(U1, TP1, {CDI1, CDI2}) allowed

(U5, TP2, {CDI1}) not allowed

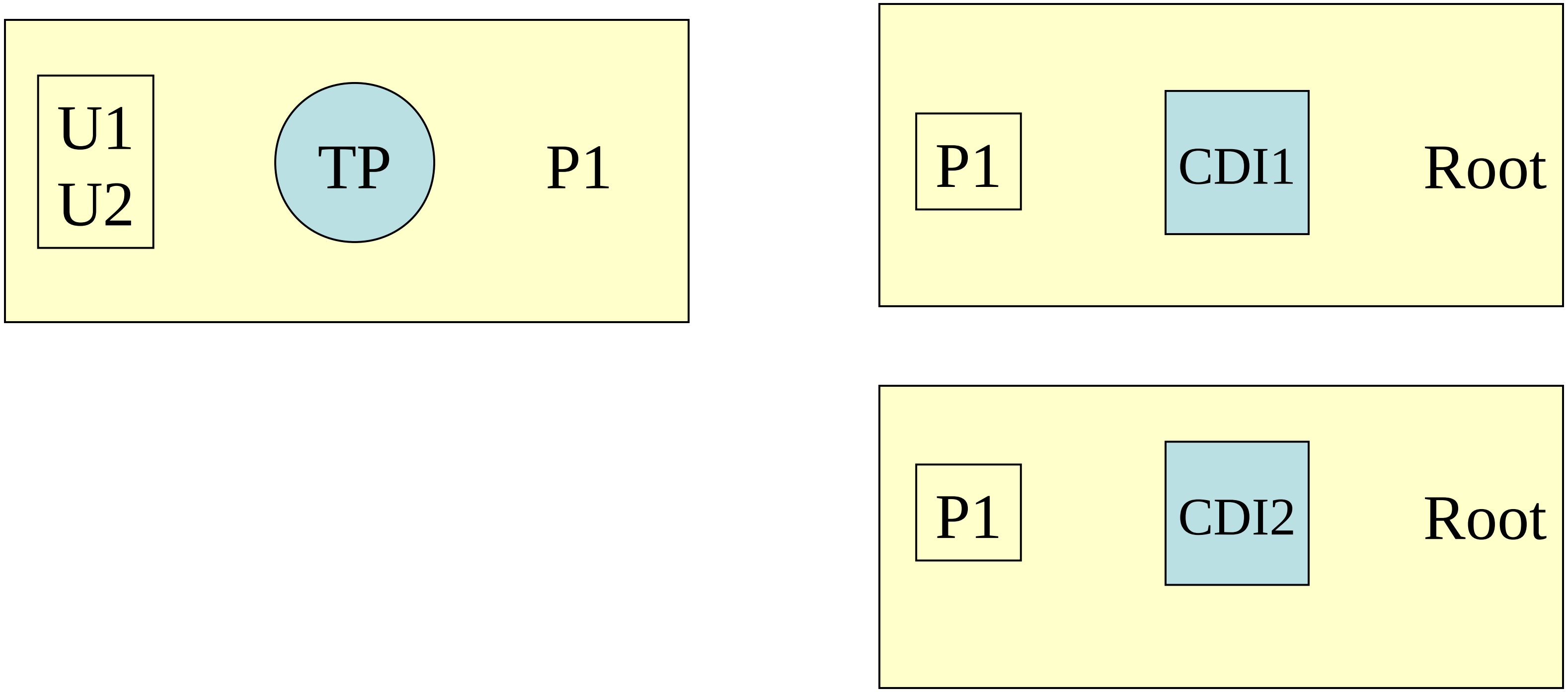
Examples

- Access to CDI constrained by user
  - In “allowed” triple, *TP* can be any TP
  - Put CDIs in a group containing all users authorized to modify CDI
- Access to CDI constrained by TP
  - In “allowed” triple, *user* can be any user
  - CDIs allow access to the owner, the user owning the TP
  - Make the TP world executable

Problem

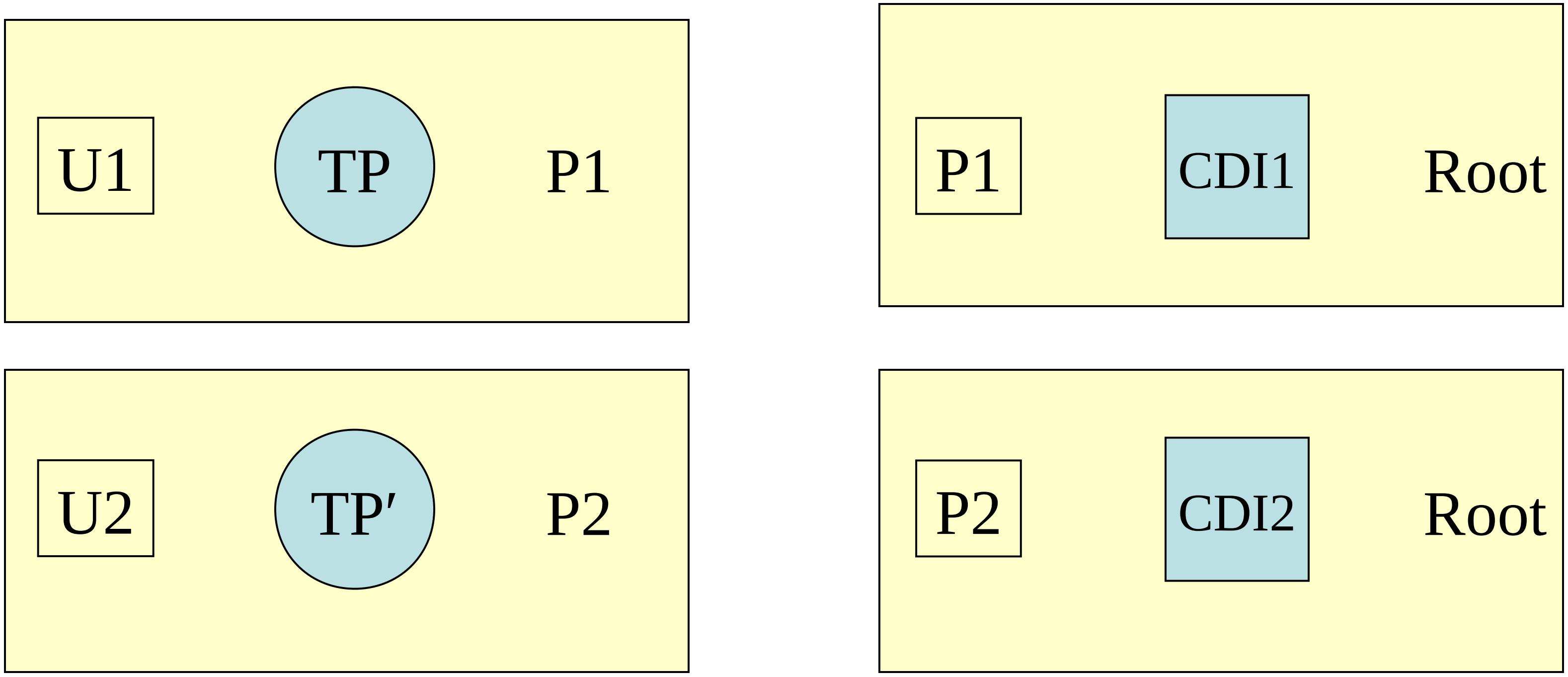
- 2 different users cannot use same copy of TP to access 2 different CDIs
  - Allow (U1, TP, {CDI1})
  - Allow (U2, TP, {CDI2})
  - Do not allow (U1, TP, {CDI2})

Problem Illustrated



Solution

Use 2 separate copies of TP1 (one for each user and CDI set)



Other Problems

- TPs are setuid programs
  - As these change privileges, want to minimize their number
- *root* can assume identity of users owning TPs, and so cannot be separated from certifiers
  - No way to overcome this without changing nature of *root*

Key Points

- Integrity policies deal with trust
  - As trust is hard to quantify, these policies are hard to evaluate completely
  - Look for assumptions and trusted users to find possible weak points in their implementation
- Biba, Lipner based on multilevel integrity
- Clark-Wilson focuses on separation of duty and transactions