
Trusted Models

CS461/ECE422

Fall 2007

Reading

- Chapter 5.1 – 5.3 (stopping at “Models Proving Theoretical Limitations”) in *Security in Computing*

Outline

- Trusted System Basics
- Specific Policies and models
 - Military Policy
 - Bell-LaPadula Model
 - Commercial Policy
 - Biba Model
 - Separation of Duty
 - Clark-Wilson
 - Chinese Wall

What is a Trusted System?

- Correct implementation of critical features
 - Features
 - Assurance
 - Personal evaluation
 - Review in the paper or on key web site
 - Friend's recommendation
 - Marketing literature

Some Key Characteristics of Trusted Systems

- Functional Correctness
- Enforcement of Integrity
- Limited Privilege
- Appropriate Confidence

MAC vs DAC

- Discretionary Access Control (DAC)
 - Normal users can change access control state directly assuming they have appropriate permissions
 - Access control implemented in standard OS's, e.g., Unix, Linux, Windows
 - Access control is at the discretion of the user
- Mandatory Access Control (MAC)
 - Access decisions cannot be changed by normal rules
 - Generally enforced by system wide set of rules
 - Normal user cannot change access control schema
- “Strong” system security requires MAC
 - Normal users cannot be trusted

Military or Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
 - Need-to-Know
 - 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	Ulaley	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Ulaley 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
- Meaning of “secure” in axiomatic

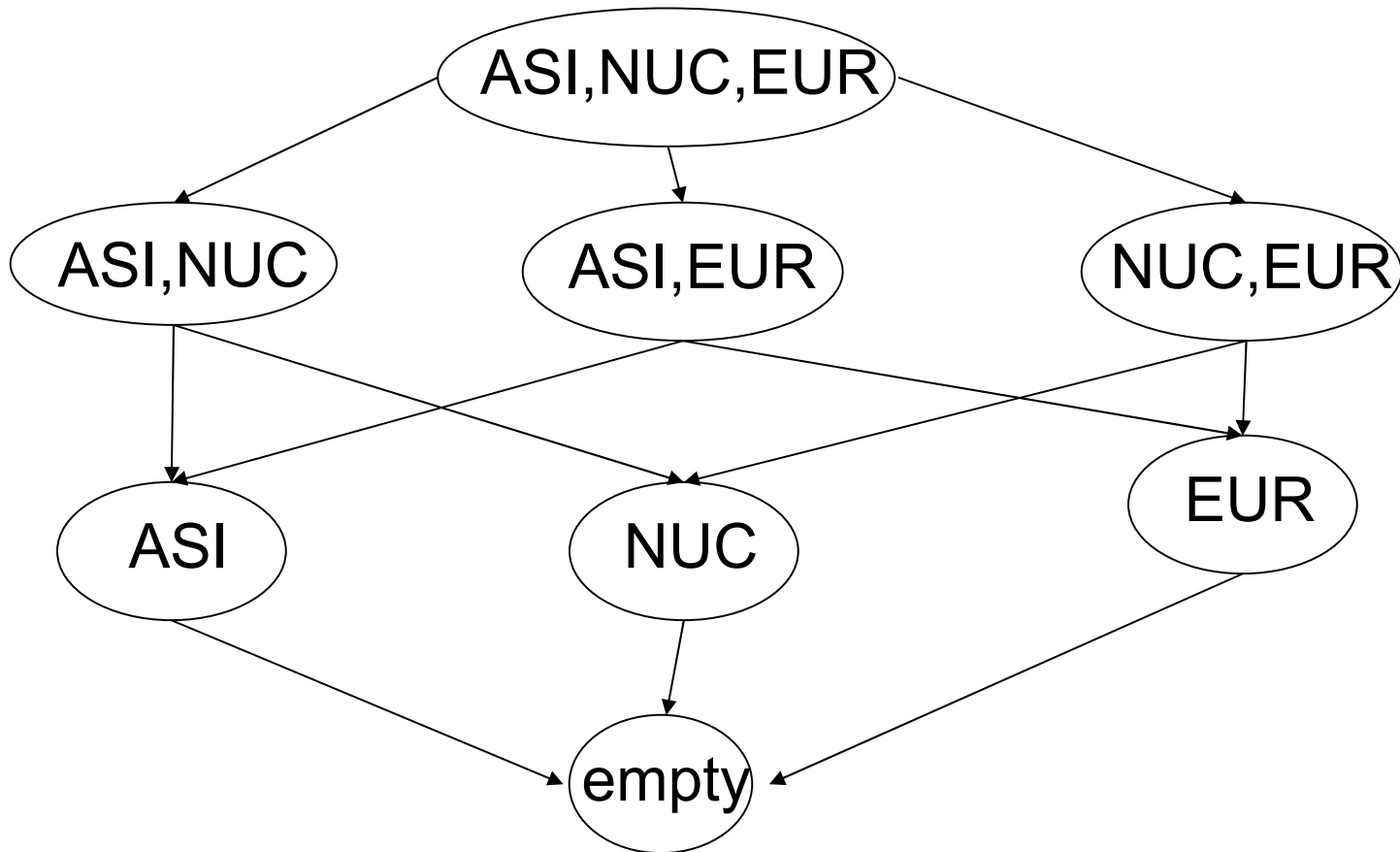
Bell-LaPadula Model, Step 2

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

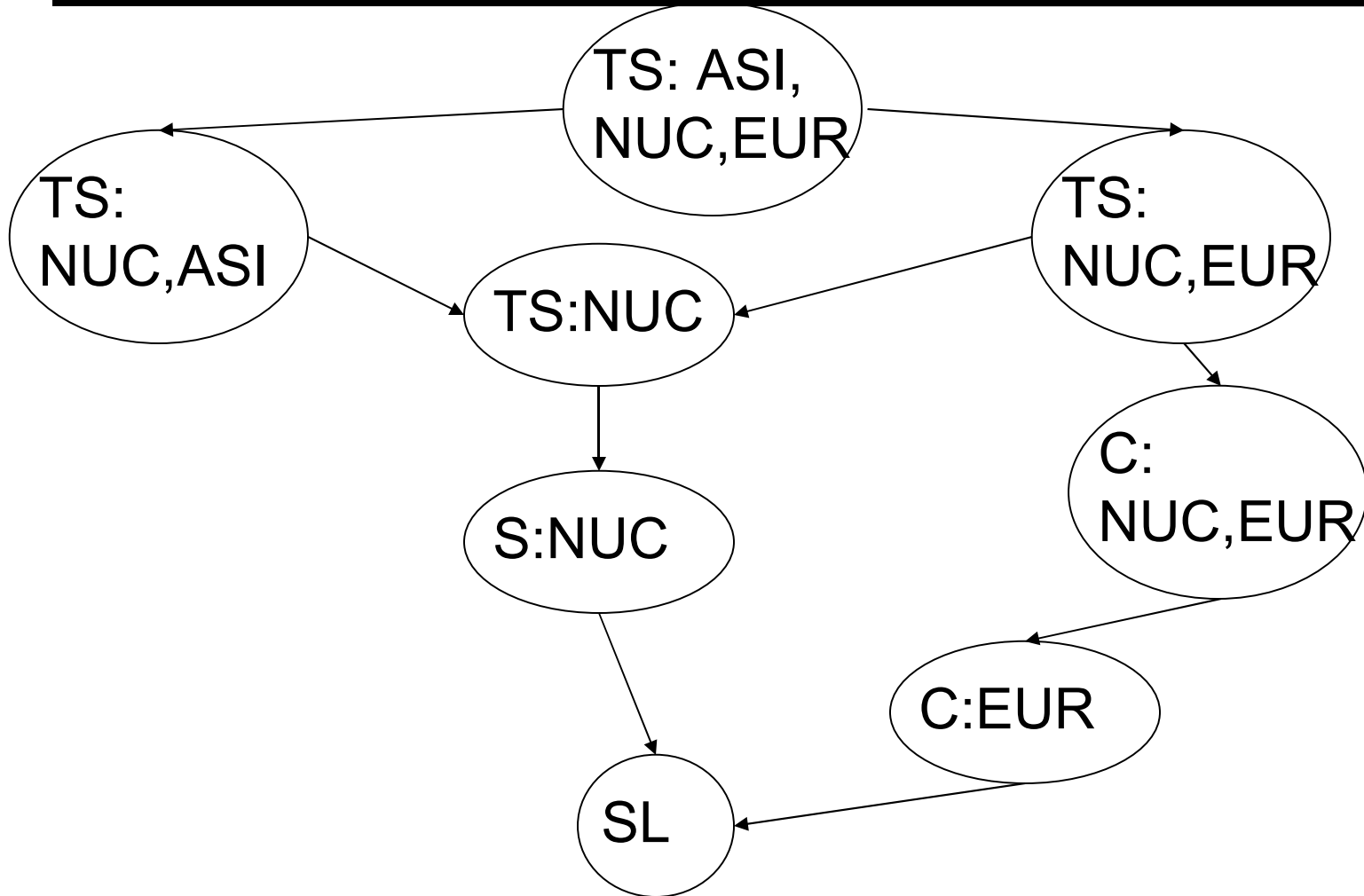
Levels and Lattices

- $(A, C) \text{ dom } (A', C')$ iff $A' \leq A$ and $C' \subseteq C$
- Examples
 - (Top Secret, {NUC, ASI}) dom (Secret, {NUC})
 - (Secret, {NUC, EUR}) dom (Confidential, {NUC, EUR})
 - (Top Secret, {NUC}) $\neg \text{dom}$ (Confidential, {EUR})
 - (Secret, {NUC}) $\neg \text{dom}$ (Confidential, {NUC, EUR})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, dom form lattice
 - *Partially ordered set*
 - *Any pair of elements*
 - *Has a greatest lower bound*
 - *Has a least upper bound*

Example Lattice



Subset Lattice



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.

Problem

- Colonel has (Secret, {NUC, EUR}) clearance
- Major has (Secret, {EUR}) clearance
- Can Major write data that Colonel can read?
- Can Major read data that Colonel wrote?
- What about the reverse?

Solution

- Define maximum, current levels for subjects
 - $maxlevel(s) \text{ dom } curlevel(s)$
- Example
 - Treat Major as an object (Colonel is writing to him/her)
 - Colonel has $maxlevel$ (Secret, { NUC, EUR })
 - Colonel sets $curlevel$ to (Secret, { EUR })
 - Now $L(\text{Major}) \text{ dom } curlevel(\text{Colonel})$
 - Colonel can write to Major without violating “no writes down”
 - Does $L(s)$ mean $curlevel(s)$ or $maxlevel(s)$?
 - Formally, we need a more precise notation

Adjustments to “write up”

- General write permission is both read and right
 - So both simple security condition and *-property apply
 - $S \text{ dom } O$ and $O \text{ dom } S$ means $S=O$
- BLP discuss append as a “pure” write so writeup still applies

Principle of Tranquillity

- Raising object's security level
 - Information once available to some subjects is no longer available
 - Usually assume information has already been accessed, so this does nothing
- Lowering object's security level
 - The *declassification problem*
 - Essentially, a “write down” violating *-property
 - Solution: define set of trusted subjects that *sanitize* or remove sensitive information before security level lowered

Types of Tranquillity

- Strong Tranquillity
 - The clearances of subjects, and the classifications of objects, do not change during the lifetime of the system
- Weak Tranquillity
 - The clearances of subjects, and the classifications of objects change in accordance with a specified policy.

Example

- DG/UX System
 - Only a trusted user (security administrator) can lower object's security level
 - In general, process MAC labels cannot change
 - If a user wants a new MAC label, needs to initiate new process
 - Cumbersome, so user can be designated as able to change process MAC label within a specified range
- Other systems allow multiple labeled windows to address users operating a multiple levels

Commercial Policies

- Less hierarchical than military
 - More dynamic
- Concerned with integrity and availability in addition to confidentiality

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.

Biba Integrity Model

Basis for all 3 models:

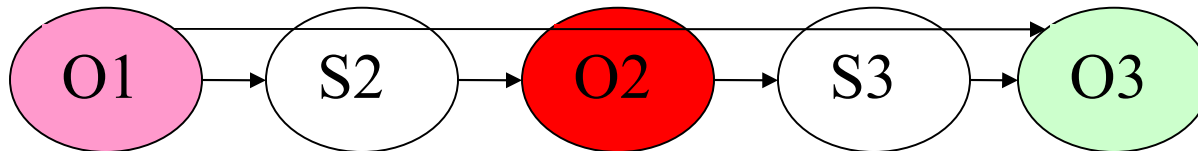
- Set of subjects S , objects O , integrity levels I , relation $\leq \subseteq I \times I$ holding when second dominates first
- $\text{min}: I \times I \rightarrow I$ returns lesser of integrity levels
- $i: S \cup O \rightarrow I$ gives integrity level of entity
- $\underline{r} \subseteq S \times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

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, \dots, o_{n+1} and corresponding sequence of subjects s_1, \dots, s_n such that $s_i \underline{r} o_i$ and $s_i \underline{w} o_{i+1}$ for all $i, 1 \leq i \leq n$.
- Idea: information can flow from o_1 to o_{n+1} along this path by successive reads and writes

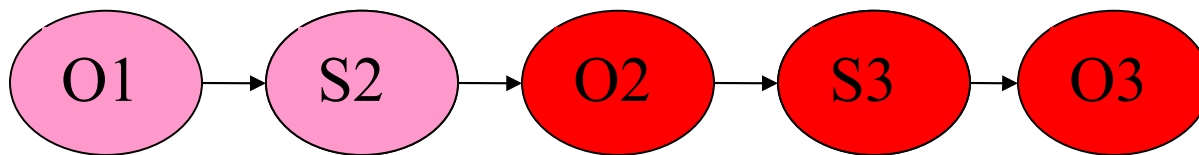


Low-Water-Mark Policy

- 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 $i(o) \leq i(s)$.
 - 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 $i(s_2) \leq i(s_1)$.

Information Flow and Model

- If there is information transfer path from $o_1 \in O$ to $o_{n+1} \in O$, enforcement of low-watermark 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 $i(o) \leq i(s)$.
 - Any subject can read any object.
 - $s_1 \in S$ can execute $s_2 \in S$ if and only if $i(s_2) \leq i(s_1)$.
- 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 $i(s) \leq i(o)$
 - $s \in S$ can write to $o \in O$ iff $i(o) \leq i(s)$
 - $s_1 \in S$ can execute $s_2 \in O$ iff $i(s_2) \leq i(s_1)$
- 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
- Implemented today as Mandatory Integrity Controls (MIC)

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

Separation of Duty Policy

- If the same individual holds multiple roles, conflict of interest may result
- Example:
 - Issue Order
 - Receive Order
 - Pay Order

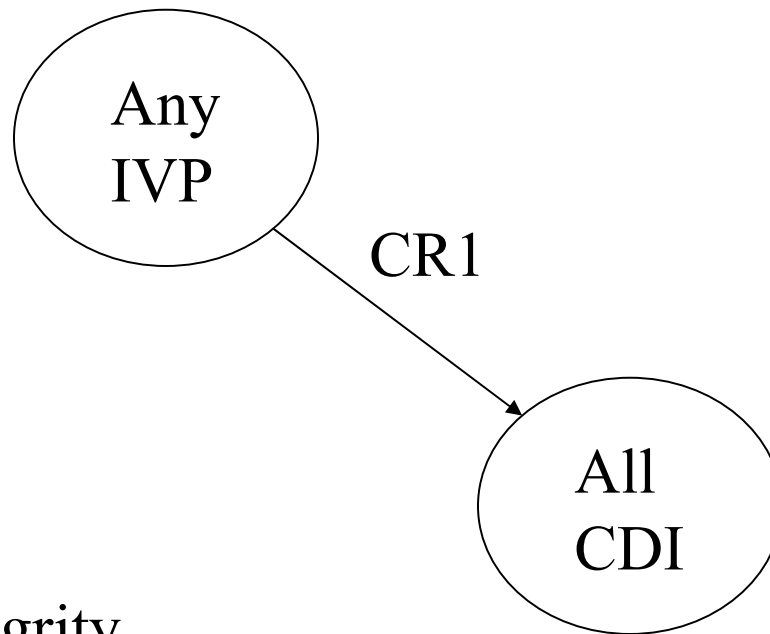
Clark-Wilson Integrity Model

- 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?

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

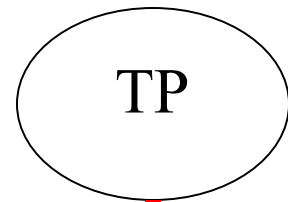
Certification Rule 1



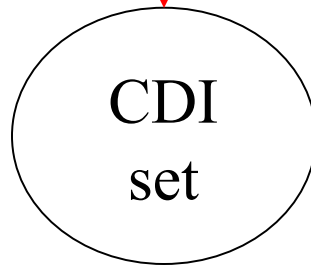
CDI = Constrained
Data Item

IVP = Integrity
Verification Procedure

CR1 and ER2: Certified Relationship



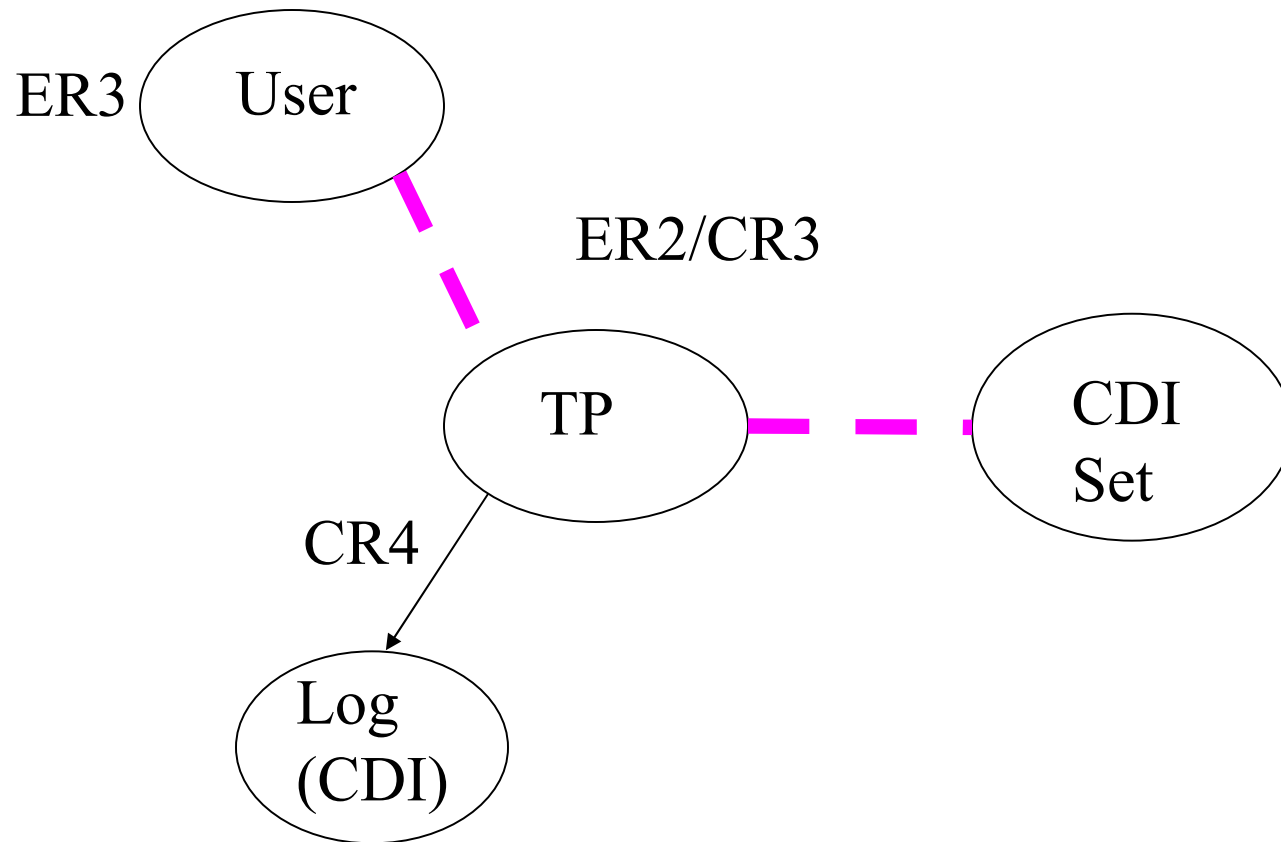
CR2
ER2



CDI = Constrained
Data Item

TP = Transaction
Procedure

Allowed Relationship and Logging



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)

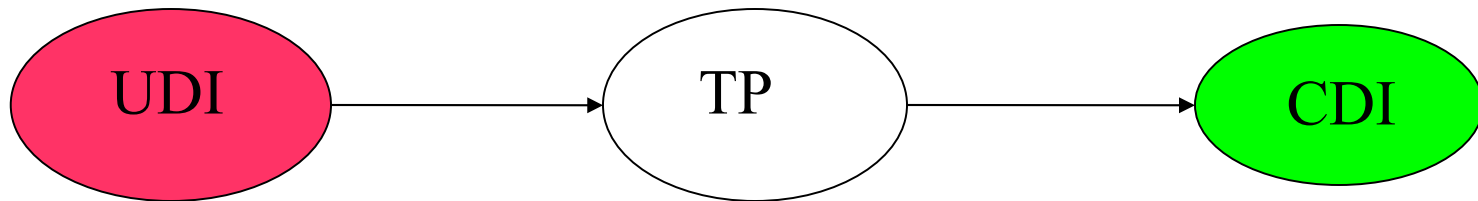
Users and Rules

- 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

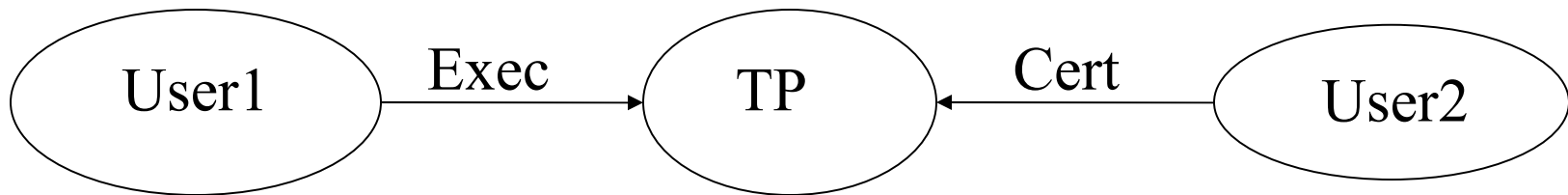


UDI = Unconstrained Data Item

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)

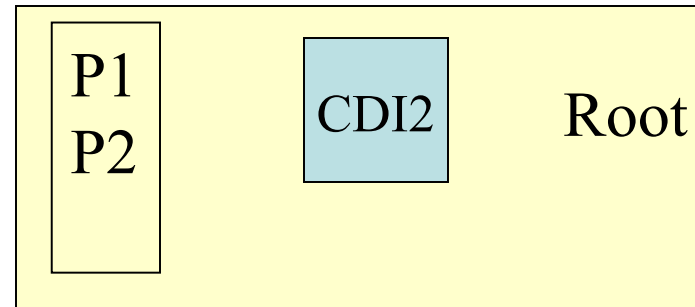
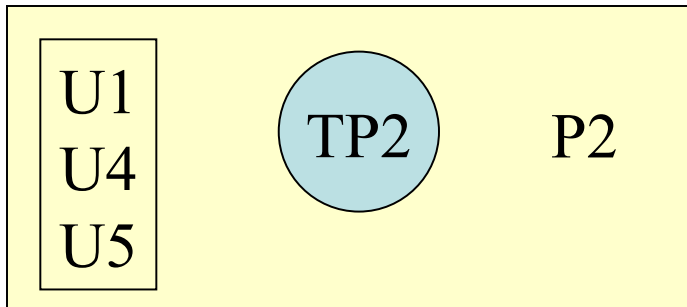
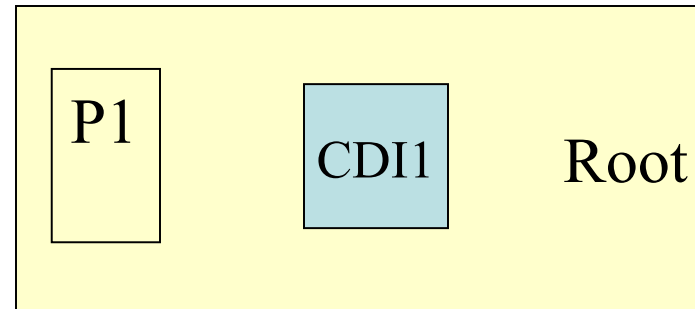
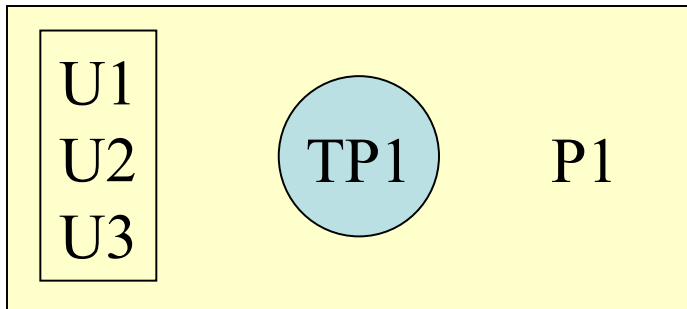
UNIX Implementation

- 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

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

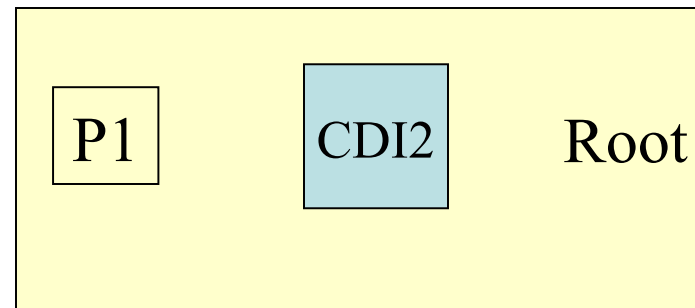
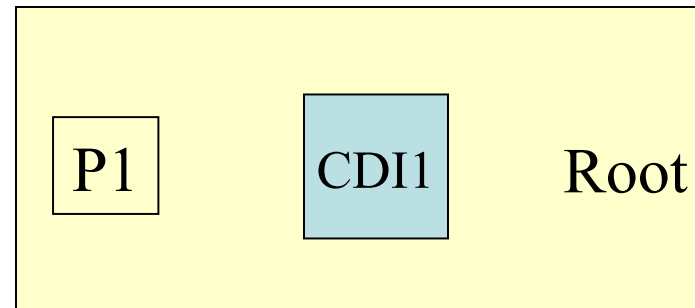
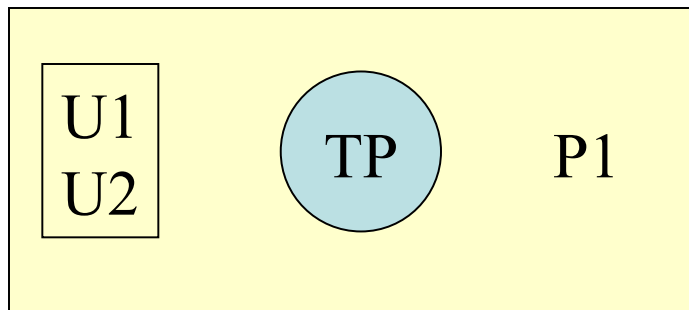
Examples

- 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

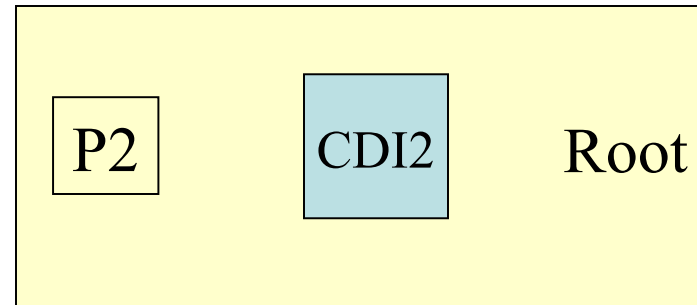
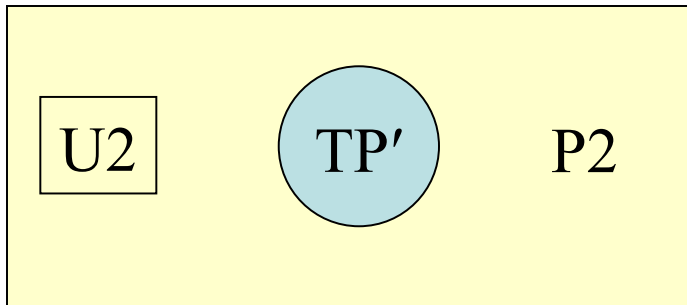
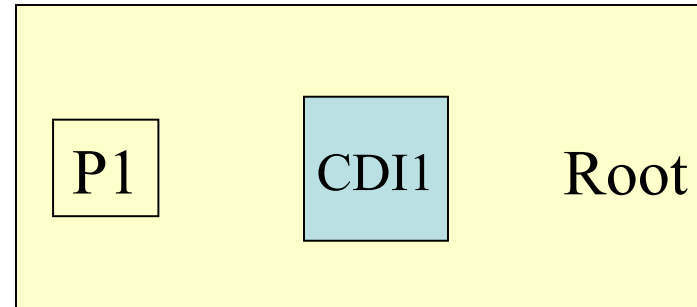
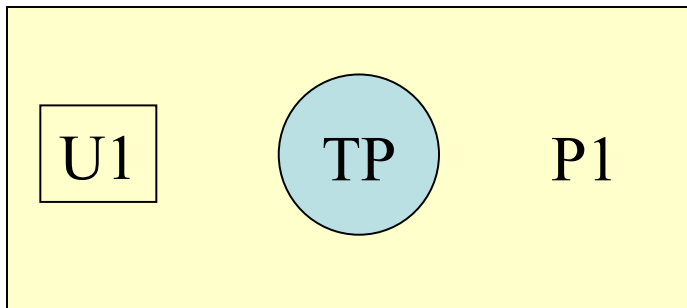
- 2 different users cannot use same copy of TP to access 2 different CDIs
 - Allow (U1, TP, {CDI1})
 - Allow (U2, TP, {CDI2})
 - Do not allow (U1, TP, {CDI2})

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*

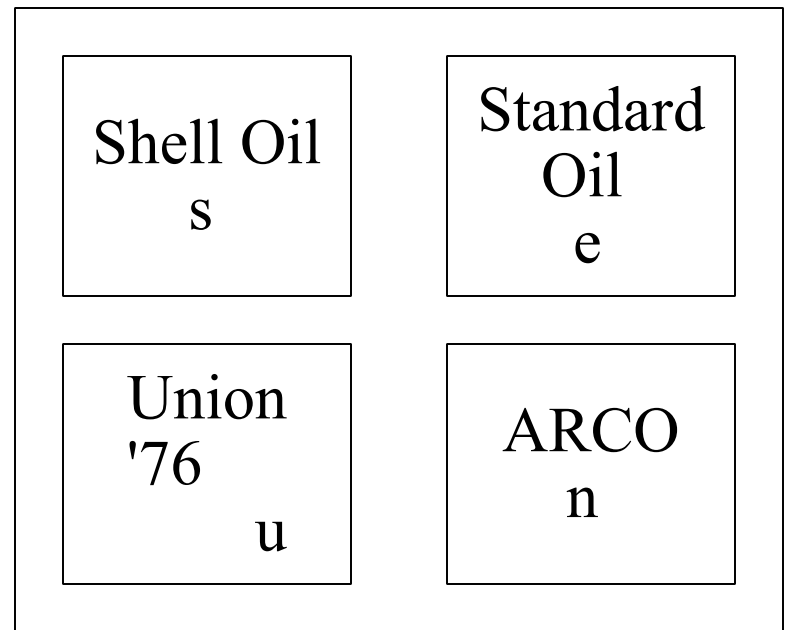
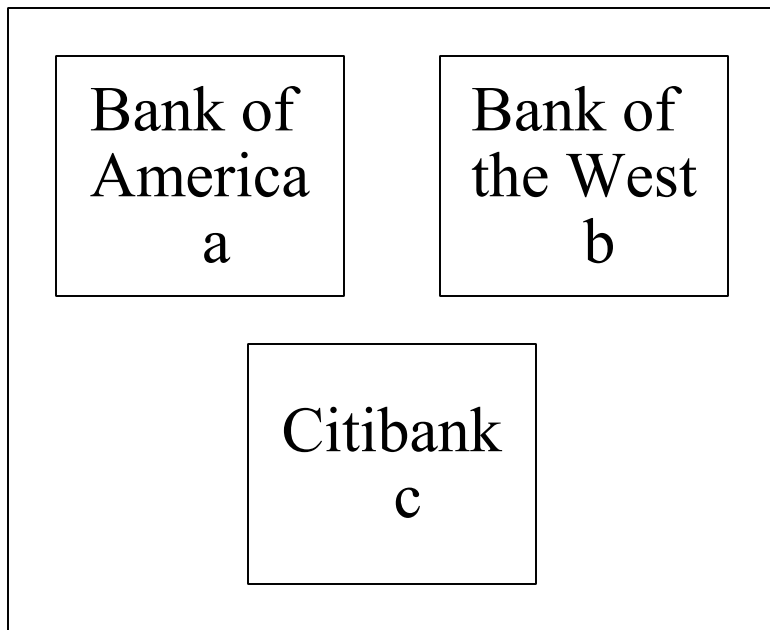
Chinese Wall

- Another way of dealing with Conflict of Interest
- Term used in banking circles since late 1920's
 - Broker may serve two clients whose interests conflict
 - Energy company may both bid on energy on the market and produce energy for the market
- Brewer and Nash developed a formal CS model in 1989

Definitions

- Objects – Files or DB elements accessed
- Company Group or Company Dataset (CD) – Set of objects concerning a particular company
- Conflict Class or Conflict of Interest Class (COI) – Set of companies that operate in the same area or otherwise have conflicting interests

Example

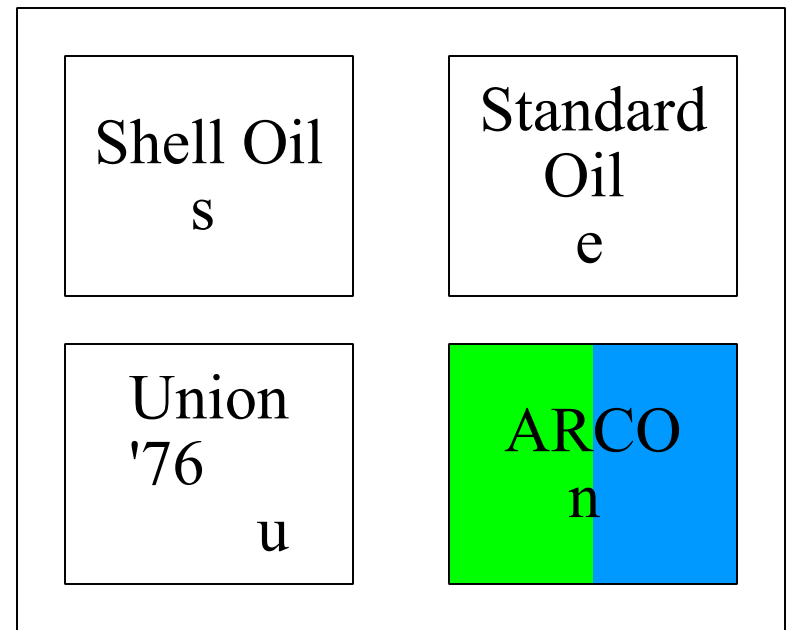
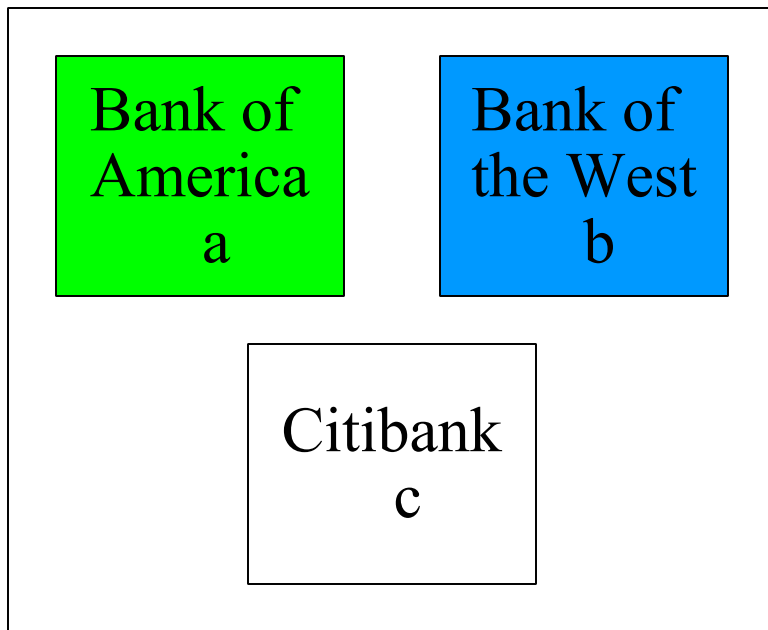


CW-Simple Security Policy

- S can read O if and only if either of the following is true
 - There is an object O' s.t. S has accessed O' and $CD(O') = CD(O)$
 - For all objects O', O' element of PR(S) implies $COI(O') \neq COI(O)$
- PR(S) is the set of all objects read by S since the beginning of time

Write Issue?

Bob Green and Alice Blue



CW *-Property

- A subject S may write an object O if and only if both of the following conditions hold
 - The CW-simple security condition permits S to read O .
 - For all unsanitized objects O' , S can read O' implies $CD(O') = CD(O)$

Key Points

- Trust vs Security
 - Assurance
- Classic Trust Policies and Models
 - Address Confidentiality and Integrity to varying degrees