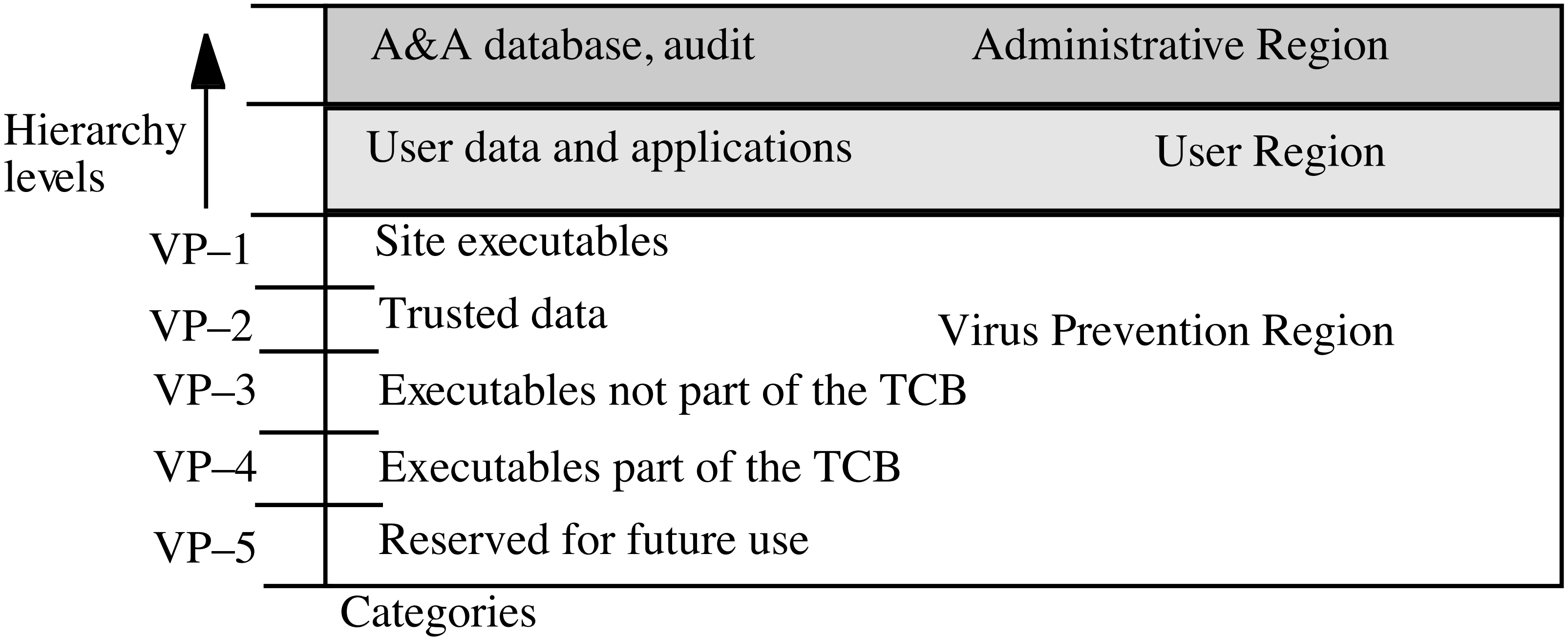
DG/UX System

- Provides mandatory access controls
  - MAC label identifies security level
  - Default labels, but can define others
- Initially
  - Subjects assigned MAC label of parent
    - Initial label assigned to user, kept in Authorization and Authentication database
  - Object assigned label at creation
    - Explicit labels stored as part of attributes
    - Implicit labels determined from parent directory

MAC Regions



IMPL_HI is “maximum” (least upper bound) of all levels
IMPL_LO is “minimum” (greatest lower bound) of all levels

Directory Problem

- Process *p* at MAC_A tries to create file /*tmp*/*x*
- /*tmp*/*x* exists but has MAC label MAC_B
  - Assume MAC_B dom MAC_A
- Create fails
  - Now *p* knows a file named *x* with a higher label exists
- Fix: only programs with same MAC label as directory can create files in the directory
  - Now compilation won't work, mail can't be delivered

Multilevel Directory

- Directory with a set of subdirectories, one per label
  - Not normally visible to user
  - p creating /*tmp*/*x* actually creates /*tmp*/*d*/*x* where *d* is directory corresponding to MAC_A
  - All *p*'s references to /*tmp* go to /*tmp*/*d*
- *p* cd's to /*tmp*/*a*, then to ..
  - System call stat(".", &buf) returns inode number of real directory
  - System call dg_stat(".", &buf) returns inode of /*tmp*

Object Labels

- Requirement: every file system object must have MAC label

1. Roots of file systems have explicit MAC labels
    - If mounted file system has no label, it gets label of mount point
2. Object with implicit MAC label inherits label of parent

Object Labels

- Problem: object has two names
  - */x/y/z*, */a/b/c* refer to same object
  - *y* has explicit label IMPL_HI
  - *b* has explicit label IMPL_B
- Case 1: hard link created while file system on DG/UX system, so …

3. Creating hard link requires explicit label
   - If implicit, label made explicit
   - Moving a file makes label explicit

Object Labels

- Case 2: hard link exists when file system mounted
  - No objects on paths have explicit labels: paths have same *implicit* labels
  - An object on path acquires an explicit label: implicit label of child must be preserved

  so …

4. Change to directory label makes child labels explicit *before* the change

Object Labels

- Symbolic links are files, and treated as such, so …

5. When resolving symbolic link, label of object is label of target of the link
    - System needs access to the symbolic link itself

Using MAC Labels

- Simple security condition implemented
- *-property not fully implemented
  - Process MAC must equal object MAC
  - Writing allowed only at same security level
- Overly restrictive in practice

MAC Tuples

- Up to 3 MAC ranges (one per region)
- MAC range is a set of labels with upper, lower bound
  - Upper bound must dominate lower bound of range
- Examples
  1. [(Secret, {NUC}), (Top Secret, {NUC})]
  2. [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
  3. [(Confidential, {ASI}), (Secret, {NUC, ASI})]

MAC Ranges

1. [(Secret, {NUC}), (Top Secret, {NUC})]
2. [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
3. [(Confidential, {ASI}), (Secret, {NUC, ASI})]

- (Top Secret, {NUC}) in ranges 1, 2
- (Secret, {NUC, ASI}) in ranges 2, 3
- [(Secret, {ASI}), (Top Secret, {EUR})] not valid range
  - as (Top Secret, {EUR}) ¬*dom* (Secret, {ASI})

Objects and Tuples

- Objects must have MAC labels
  - May also have MAC label
  - If both, tuple overrides label
- Example
  - Paper has MAC range:

    [(Secret, {EUR}), (Top Secret, {NUC, EUR})]

MAC Tuples

- Process can read object when:
  - Object MAC range (*lr*, *hr*); process MAC label *pl*
  - *pl dom hr*
    - Process MAC label grants read access to upper bound of range
- Example
  - Peter, with label (Secret, {EUR}), cannot read paper
    - (Top Secret, {NUC, EUR}) *dom* (Secret, {EUR})
  - Paul, with label (Top Secret, {NUC, EUR, ASI}) can read paper
    - (Top Secret, {NUC, EUR, ASI}) *dom* (Top Secret, {NUC, EUR})

MAC Tuples

- Process can write object when:
  - Object MAC range (*lr*, *hr*); process MAC label *pl*
  - $pl \in (lr, hr)$
    - Process MAC label grants write access to any label in range
- Example
  - Peter, with label (Secret, {EUR}), can write paper
    - (Top Secret, {NUC, EUR}) *dom* (Secret, {EUR}) and (Secret, {EUR}) *dom* (Secret, {EUR})
  - Paul, with label (Top Secret, {NUC, EUR, ASI}), cannot read paper
    - (Top Secret, {NUC, EUR, ASI}) *dom* (Top Secret, {NUC, EUR})

Principle of Tranquility

- 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 Tranquility

- Strong Tranquility
  - The clearances of subjects, and the classifications of objects, do not change during the lifetime of the system
- Weak Tranquility
  - The clearances of subjects, and the classifications of objects, do not change in a way that violates the simple security condition or the *-property during the lifetime of the system

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

Overview

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

Requirements of 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

- Model defines integrity levels analogous to Bell-LaPadula Model’s security levels
- Set of subjects *S*, objects *O*, integrity levels *I*
- Relation $a \le b$ holding when second integrity level dominates first
- $i(a)$ is integrity level of entity

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*

Strict Integrity Policy

- 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 $i(s_2) \leq i(s_1)$
- 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

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

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

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
5. 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

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

Problems

- 2 different users cannot use same copy of TP to access 2 different CDIs
  - Need 2 separate copies of TP (one for each user and CDI set)
- 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 based on multilevel integrity
- Clark-Wilson focuses on separation of duty and transactions