

# Lecture 5

## September 30, 2022

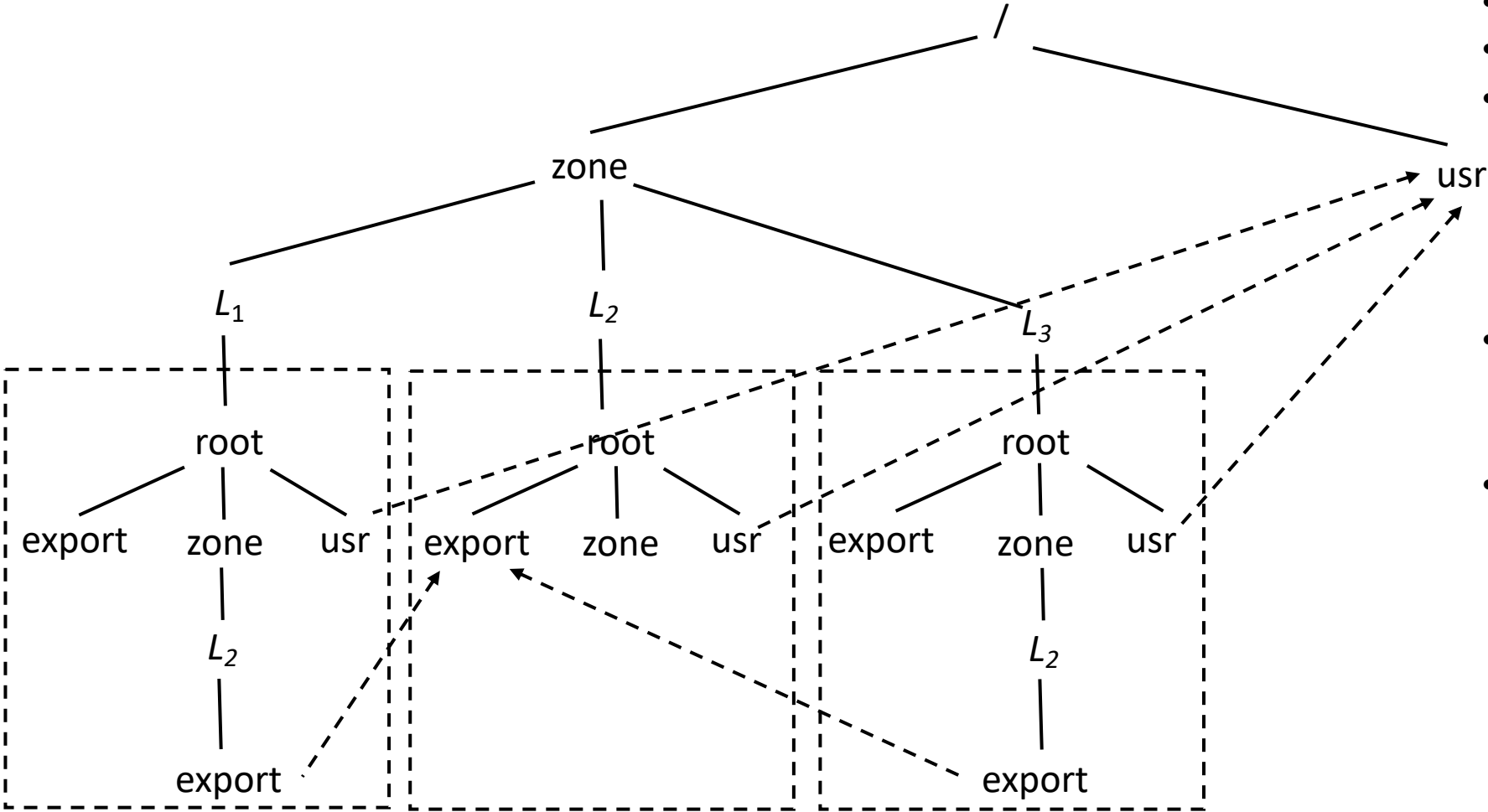
# Labeled Zones

- Used in Trusted Solaris Extensions, various flavors of Linux
- *Zone*: virtual environment tied to a unique label
  - Each process can only access objects in its zone
- *Global zone* encompasses everything on system
  - Its label is ADMIN\_HIGH
  - Only system administrators can access this zone
- Each zone has a unique root directory
  - All objects within the zone have that zone's label
  - Each zone has a unique label

# More about Zones

- Can import (mount) file systems from other zones provided:
  - If importing *read-only*, importing zone's label must dominate imported zone's label
  - If importing *read-write*, importing zone's label must equal imported zone's label
    - So the zones are the same; import unnecessary
  - Labels checked at time of import
- Objects in imported file system retain their labels

# Example



- $L_1 \text{ dom } L_2$
- $L_3 \text{ dom } L_2$
- Process in  $L_1$  can read any file in the export directory of  $L_2$  (assuming discretionary permissions allow it)
- $L_1, L_3$  disjoint
  - Do not share any files
- System directories imported from global zone, at ADMIN\_LOW
  - So can only be read

# 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: Trusted Solaris

- Security administrator can provide specific authorization for a user to change the MAC label of a file
  - “downgrade file label” authorization
  - “upgrade file label” authorization
- User requires additional authorization if not the owner of the file
  - “act as file owner” authorization

# 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.



# Principles of Operation

- *Separation of duty*: if two or more steps are required to perform a critical function, at least two different people should perform the steps
- *Separation of function*: different entities should perform different functions
- *Auditing*: recording enough information to ensure the abilities to both recover and determine accountability

# 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
- $\underline{r}: S \times O$  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

# Strict Integrity Policy

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

# Principles of Declassification

- Principle of Semantic Consistency
  - As long as semantics of components that do not do declassification do not change, the components can be altered without affecting security
- Principle of Occlusion
  - A declassification operation cannot conceal an *improper* declassification
- Principle of Conservativity
  - Absent any declassification, the system is secure
- Principle of Monotonicity of Release
  - When declassification is performed in an authorized manner by authorized subjects, the system remains secure

# 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)