Lecture #11

- Bell-LaPadula model
  - Weak Tranquility
  - System Z and the controversy
- Requirements
  - Very different than confidentiality policies
- Biba's models
- Clark-Wilson model

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 of Weak Tranquility

- Only one subject at TOP SECRET
- Document at CONFIDENTIAL
- New CONFIDENTIAL user to be added
  - User should not see document
- Raise document to SECRET
  - Subject still cannot write document
  - All security relationships unchanged

Controversy

- McLean:
  - "value of the BST is much overrated since there is a great deal more to security than it captures. Further, what is captured by the BST is so trivial that it is hard to imagine a realistic security model for which it does not hold."
  - Basis: given assumptions known to be non-secure, BST can prove a non-secure system to be secure

†-Property

- State (b, m, f, h) satisfies the †-property iff for each $s \in S$ the following hold:
  1. $b(s: \underline{\text{a}}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{\text{a}})\ [\ f_c(s)\ dom\ f_o(o)\ ]\ ]$
  2. $b(s: \underline{\text{w}}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{\text{w}})\ [\ f_o(o) = f_c(s)\ ]\ ]$
  3. $b(s: \underline{\text{r}}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{\text{r}})\ [\ f_c(s)\ dom\ f_o(o)\ ]\ ]$
- Idea: for writing, subject dominates object; for reading, subject also dominates object
- Differs from *-property in that the mandatory condition for writing is reversed
  - For *-property, it's object dominates subject

Analogues

The following two theorems can be proved

- $\Sigma(R, D, W, z_0)$ satisfies the †-property relative to $S' \subseteq S$ for any secure state z_0 iff for every action $(r, d, (b, m, f, h), (b', m', f', h'))$, W satisfies the following for every $s \in S'$
  - Every $(s, o, p) \in b - b'$ satisfies the †-property relative to S'
  - Every $(s, o, p) \in b'$ that does not satisfy the †-property relative to S' is not in b
- $\Sigma(R, D, W, z_0)$ is a secure system if z_0 is a secure state and W satisfies the conditions for the simple security condition, the †-property, and the ds-property.

Problem

- This system is *clearly* non-secure!
  - Information flows from higher to lower because of the †-property

Discussion

- Role of Basic Security Theorem is to demonstrate that rules preserve security
- Key question: what is security?
  - Bell-LaPadula defines it in terms of 3 properties (simple security condition, *-property, discretionary security property)
  - Theorems are assertions about these properties
  - Rules describe changes to a *particular* system instantiating the model
  - Showing system is secure requires proving rules preserve these 3 properties

Rules and Model

- Nature of rules is irrelevant to model
- Model treats “security” as axiomatic
- Policy defines “security”
  - This instantiates the model
  - Policy reflects the requirements of the systems
- McLean’s definition differs from Bell-LaPadula
  - … and is not suitable for a confidentiality policy
- Analysts cannot prove “security” definition is appropriate through the model

System Z

- System supporting weak tranquility
- On *any* request, system downgrades *all* subjects and objects to lowest level and adds the requested access permission
  - Let initial state satisfy all 3 properties
  - Successive states also satisfy all 3 properties
- Clearly not secure
  - On first request, everyone can read everything

Reformulation of Secure Action

- Given state that satisfies the 3 properties, the action transforms the system into a state that satisfies these properties and eliminates any accesses present in the transformed state that would violate the property in the initial state, then the action is secure
- BST holds with these modified versions of the 3 properties

Reconsider System Z

- Initial state:
  - subject s, object o
  - $C = \{\text{High}, \text{Low}\}$, $K = \{\text{All}\}$
- Take:
  - $f_c(s) = (\text{Low}, \{\text{All}\})$, $f_o(\text{o}) = (\text{High}, \{\text{All}\})$
  - $m[s, o] = \{ \underline{\text{w}} \}$, and $b = \{ (s, o, \underline{\text{w}}) \}$.
- s requests $\underline{\text{r}}$ access to o
- Now:
  - $f'_o(o) = (\text{Low}, \{\text{All}\})$
  - $(s, o, \underline{\text{r}}) \in b'$, $m'[s, o] = \{\underline{\text{r}}, \underline{\text{w}}\}$

Non-Secure System Z

- As $(s, o, \underline{\text{r}}) \in b' - b$ and $f_o(o)$ *dom* $f_c(s)$, access added that was illegal in previous state
  - Under the new version of the Basic Security Theorem, System Z is not secure
  - Under the old version of the Basic Security Theorem, as $f'_c(s) = f'_o(o)$, System Z is secure

Response: What Is Modeling?

- Two types of models
    1. Abstract physical phenomenon to fundamental properties
    2. Begin with axioms and construct a structure to examine the effects of those axioms
- Bell-LaPadula Model developed as a model in the first sense
    - McLean assumes it was developed as a model in the second sense

Reconciling System Z

- Different definitions of security create different results
  - Under one (original definition in Bell-LaPadula Model), System Z is secure
  - Under other (McLean's definition), System Z is not secure

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

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
- $\underline{\text{r}}$: $S \times O$ means $s \in S$ can read $o \in O$
- $\underline{\text{w}}$, $\underline{\text{x}}$ 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 $o_1, \ldots, o_{n+1}$ and corresponding sequence of subjects $s_1, \ldots, s_n$ such that $s_i \; \underline{\mathrm{r}} \; o_i$ and $s_i \; \underline{\mathrm{w}} \; o_{i+1}$ 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

  1. $s \in S$ can write to $o \in O$ if and only if $i(o) \leq i(s)$.
  2. 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.
  3. $s_1 \in S$ 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-water-mark policy requires $i(o_{n+1}) \le i(o_1)$ for all $n > 1$.
  - Idea of proof: Assume information transfer path exists between o_1 and o_{n+1}. Assume that each read and write was performed in the order of the indices of the vertices. By induction, the integrity level for each subject is the minimum of the integrity levels for all objects preceding it in path, so $i(s_n) \le i(o_1)$. As *n*th write succeeds, $i(o_{n+1}) \le i(s_n)$. Hence $i(o_{n+1}) \le i(o_1)$.

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
    1. $s \in S$ can write to $o \in O$ if and only if $i(o) \leq i(s)$.
    2. Any subject can read any object.
    3. $s_1 \in S$ 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

- Similar to Bell-LaPadula model
  1. $s \in S$ can read $o \in O$ iff $i(s) \le i(o)$
  2. $s \in S$ can write to $o \in O$ iff $i(o) \le i(s)$
  3. $s_1 \in S$ can execute $s_2 \in S$ 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

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, Lipner based on multilevel integrity
- Clark-Wilson focuses on separation of duty and transactions