ECS 289M Lecture 10

April 21, 2006

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 $o_1, ..., o_{n+1}$ and corresponding sequence of subjects $s_1, ..., s_n$ such that s_i r o_i and s_i 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 $i(o_{n+1}) \leq i(o_1)$ 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 $i(s_n) \leq i(o_1)$. As *n*th write succeeds, $i(o_{n+1}) \leq i(s_n)$. Hence $i(o_{n+1}) \leq i(o_1)$.

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 $i(o) \le i(s)$.
  2. Any subject can read any object.
  3. $s_1 \in S$ can execute $s_2 \in S$ if and only if $i(s_2) \le i(s_1)$.
- 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

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

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 audited
5. System management and audit users are in AM and so have access to system state and logs

Problem

- Too inflexible
  - System managers cannot run programs for repairing inconsistent or erroneous production database
    - System managers at AM, production data at SL
- 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, { IP, ID})
System controllers	(SL, { SP, SD }) 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, { IP})
Production code	(SL, { SP })	(IO, { IP })
Production data	(SL, { SP })	(ISL, { IP })
Software tools	(SL, ∅)	(IO, { ID })
System programs	(SL, ∅)	(ISP, { IP, ID })
System programs in modification	(SL, { SSD })	(ISL, { ID })
System and application logs	(AM, { *appropriate* })	(ISL, ∅)
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: $D + YB - W = TB$
- *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)