Integrity Policies

CS691 – Chapter 6 of Matt Bishop

Integrity

Problem area: systems require data to be changed accurately and follow the rules. Disclosure is not a major concern.

Lipner [636] identifies five requirements for preserving data integrity:
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 nonproduction 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.

Auditing: the process of analyzing systems to determine what actions took place and who performed them. It uses extensive logging.

These requirement suggest 3 principles of operation:

- Separation of duty (two different people? perform two critical steps)
- Separation of function (program not developed on production system; production data for development needs to be sanitized.)
- Auditing. (Commercial systems emphasize recovery and accountability.)
Different Needs

- Commercial firms grant access based on individual needs and has a larger categories \( \rightarrow \) large number of security levels.
- In military environment, creation of compartment is centralized. In commercial firms, it is decentralized.
- Aggregating distributed inoncuous info, one can often deduce sensitive information. The Bell-LaPadula Model lack capability to track what questions have been asked.

Biba Integrity Model

- In 1977, Biba [94] studied the nature of the integrity of systems. He proposed three policies, one of which was the mathematical dual of the Bell-LaPadula Model.
- A system consists of a set \( S \) of subjects, a set \( O \) of objects, and a set \( I \) of integrity levels. The levels are ordered.
- The relation \( < \subseteq I \times I \) holds when the second integrity level dominates the first.
- The relation \( \leq \subseteq I \times I \) holds when the second integrity level either dominates or is the same as the first.
- The function \( \text{min}: I \times I \rightarrow I \) gives the lesser of the two integrity levels.
- The function \( i: S \cup O \rightarrow I \) returns the integrity level of an object or a subject.
- The relation \( \leq \subseteq S \times O \) defines the ability of a subject to read an object;
- the relation \( \leq \subseteq S \times O \) defines the ability of a subject to write to an object;
- the relation \( \leq \subseteq S \times S \) defines the ability of a subject to invoke (execute) another subject.
Intuition Behind Model Construction

- The higher the level, the more confidence one has that a program will execute correctly (or detect problems with its inputs and stop executing).
- Data at a higher level is more accurate, reliable, trustworthy than data at a lower level.
- Integrity labels, in general, are not also security labels. They are assigned and maintained separately, because the reasons behind the labels are different. Security labels primarily limit the flow of information; integrity labels primarily inhibit the modification of information.
- They may overlap, however, with surprising results.

Test case: Information Transfer Path

- Biba tests his policies against the notion of an information transfer path:
- **Definition 6-1.** An *information transfer path* is a sequence of objects $o_1, ..., o_{n+1}$ and a corresponding sequence of subjects $s_1, ..., s_n$ such that $s_i \text{ r } o_i$ and $s_i \text{ w } o_{i+1}$ for all $i, 1 \leq i \leq n$.
- Intuitively, data in the object $o_1$ can be transferred into the object $o_{n+1}$ along an information flow path by a succession of reads and writes.
Low-Water-Mark Policy

- Whenever a subject accesses an object, the policy changes the integrity level of the subject to the lower of the subject and the object. Specifically:
  1. If \( S \) can write to \( O \), then \( i(o) \leq i(s) \).
  2. If \( S \) reads \( O \), then \( i'(s) = \min(i(s), i(o)) \), where \( i'(s) \) is the subject's integrity level after the read.
  3. \( S_1 \) can execute \( S_2 \) if and only if \( i(S_2) \leq i(S_1) \).
- Rule 1 prevents writing to a higher level (higher trusted). Prevent implant of incorrect or false data.
- Rule 2 assume that the subject will rely on the data with lower integrity level. Therefore his integrity level should be lowered. (Contaminating subject and actions)
- Rule 3 prevent a less trusted invoker to control the execution of more trusted subjects.

Constrains Information Transfer Path

- This low-water-mark policy constrains any information transfer path.
- Theorem 6-1. If there is an information transfer path from object \( o_n \in O \) to object \( o_{n+1} \in O \), then enforcement of the low-water-mark policy requires that \( i(o_{n+1}) \leq i(o_n) \) for all \( n > 1 \).
  - Proof: Without loss of generality, assume that each read and write was performed in the order of the indices of the vertices. By induction, for any \( 1 \leq k \leq n \),
    - \( i(s_j) = \min\{i(o_j) | 1 \leq j \leq k \} \) after \( k \) reads. As the \( n \)th write succeeds, by rule 1, \( i(o_{n+1}) < i(s_n) \). Thus, by transitivity, \( i(o_{n+1}) < i(o_n) \).
    - This policy prevents direct modifications that would lower integrity labels. It also prevents indirect modification by lowering the integrity label of a subject that reads from an object with a lower integrity level.
    - The problem with this policy is that, in practice, the subjects change integrity levels. In particular, the level of a subject is nonincreasing, which means that it will soon be unable to access objects at a high integrity level.
    - How about decrease object integrity level rather than subject integrity level?
**Ring Policy**

- The ring policy ignores the issue of indirect modification and focuses on direct modification only. This solves the problems described above. The rules are as follows.
  1. Any subject may read any object, regardless of integrity levels.
  2. \(s \in S\) can write to \(o \in O\) if and only if \(i(o) \leq i(s)\).
  3. \(s_1 \in S\) can execute \(s_2 \in S\) if and only if \(i(s_2) \prec i(s_1)\).
- The difference between this policy and the low-water-mark policy is simply that any subject can read any object. Hence, Theorem 6-1 holds for this model, too.

**Biba Model (Strict Integrity Policy)**

- This model is the dual of the Bell-LaPadula Model, and is most commonly called "Biba's model."
- Its rules are as follows.
  1. \(s \in S\) can read \(o \in O\) if and only if \(i(s) \leq i(o)\).
  2. \(s \in S\) can write to \(o \in O\) if and only if \(i(o) \leq i(s)\).
  3. \(s_1 \in S\) can execute \(s_2 \in S\) if and only if \(i(s_2) \leq i(s_1)\).
- Given these rules, Theorem 6-1 still holds, but its proof changes (see Exercise 1). Note that rules 1 and 2 imply that if both read and write are allowed, \(i(s) = i(o)\).
- *Like the low-water-mark policy, this policy prevents indirect as well as direct modification of entities without authorization. By replacing the notion of "integrity level" with "integrity compartments," and adding the notion of discretionary controls, one obtains the full dual of Bell-LaPadula.*
Example: LOCUS Distributed OS

- Pozzo and Gray [817, 818] implemented Biba's strict integrity model on the distributed operating system LOCUS [811].
- Goal: limit execution domains for each program to prevent untrusted software from altering data or other software.
- Approach: make the level of trust in software and data explicit.
- They have different classes of executable programs.
- Their credibility ratings (Biba's integrity levels) assign a measure of trustworthiness on a scale from 0 (untrusted) to \( n \) (highly trusted), depending on the source of the software.
- Trusted file systems contain only executable files with the same credibility level.
- Associated with each user (process) is a risk level that starts out set to the highest credibility level at which that user can execute.
- Users may execute programs with credibility levels at least as great as the user's risk level.
- To execute programs at a lower credibility level, a user must use the run-untrusted command. This acknowledges the risk that the user is taking.

Lipner’s Integrity Matrix Model

- Lipner combines Bell LaPadula model with Biba Model to create a model that conformed more accurately to the requirements of a commercial policy.
- For clarity, we consider the Bell-LaPadula aspects of Lipner’s model first, and then combine those aspects with Biba’s model.
- Lipner provides two security levels, in the following order (higher to lower):
  - Audit Manager (AM): system audit and management functions are at this level.
  - System Low (SL): any process can read information at this level.
- He similarly defined five categories:
  1. Development (D): production programs under development and testing, but not yet in production use
  2. Production Code (PC): production processes and programs
  3. Production Data (PD): data covered by the integrity policy
  4. System Development (SD): system programs under development, but not yet in production use
  5. Software Tools (T): programs provided on the production system not related to the sensitive or protected data
Assign Security Levels

- Assigned users to security levels based on their jobs.

<table>
<thead>
<tr>
<th>Users</th>
<th>Clearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordinary users</td>
<td>(SL, { PC, PD })</td>
</tr>
<tr>
<td>Application developers</td>
<td>(SL, { D, T })</td>
</tr>
<tr>
<td>System programmers</td>
<td>(SL, { SD, T })</td>
</tr>
<tr>
<td>System managers and auditors</td>
<td>(AM, { D, PC, PD, SD, T })</td>
</tr>
<tr>
<td>System controllers</td>
<td>(SL, { D, PC, PD, SD, T })</td>
</tr>
</tbody>
</table>

- The system objects are assigned to security levels based on who should access them.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development code/test data</td>
<td>(SL, { D, T })</td>
</tr>
<tr>
<td>Production code</td>
<td>(SL, { PC })</td>
</tr>
<tr>
<td>Production data</td>
<td>(SL, { PC, PD })</td>
</tr>
<tr>
<td>Software tools</td>
<td>(SL, { T })</td>
</tr>
<tr>
<td>System programs</td>
<td>(SL, { })</td>
</tr>
<tr>
<td>System programs in modification</td>
<td>(SL, { SD, T })</td>
</tr>
<tr>
<td>System and application logs</td>
<td>(AM, { appropriate categories })</td>
</tr>
</tbody>
</table>

Does the Model Meet 5 Requirements?

- All logs are append-only. By the -*-property, their classes must dominate those of the subjects that write to them. Hence, each log will have its own categories, but the simplest way to prevent their being compromised is to put them at a higher security level.

- Check if the users can do their jobs.
  - Ordinary users can read and execute production code \( \rightarrow (SL, \{PC,PD\}) \)ou dom (SL, \{PC\})pc simple security condition
  - Ordinary users can write production data \( \rightarrow (SL, \{PC,PD\}) \)pd dom (SL, \{PC,PD\})ou -*-properties
Checking Requirements

- Check if the model meets the 5 requirements:
  1. Because users do not have execute access to category T, they cannot write their own programs, so requirement 1 is met.
  2. Application programmers and system programmers do not have read or write access to category PD, and hence cannot access production data. If they do require production data to test their programs, the data must be downgraded from PD to D, and cannot be upgraded (because the model has no upgrade privilege). The downgrading requires intervention of system control users, which is a special process within the meaning of requirement 2. Thus, requirement 2 is satisfied.
  3. The process of installing a program requires the downgrade privilege (specifically, changing the category of the program from D to PC), which belongs only to the system control users; hence, only those users can install applications or system programs. The use of the downgrade privilege satisfies requirement 3's need for a special process.
  4. The control part of requirement 4 is met by allowing only system control users to have the downgrade privilege; the auditing part is met by requiring all downgrading to be logged.
  5. Finally, the placement of system management and audit users in AM ensures that they have access both to the system state and to system logs, so the model meets requirement 5.

Problem with Simple Lipner’s Model

- Problem:
  However, the model allows little flexibility in special-purpose software. For example, a program for repairing an inconsistent or erroneous production database cannot be application-level software. To remedy these problems, Lipner integrates his model with Biba’s model.
Lipner's Full Integrity Model

- Augment the security classifications with three integrity classifications (highest to lowest):
  - System Program (ISP): the classifications for system programs
  - Operational (IO): the classifications for production programs and development software
  - System Low (ISL): the classifications at which users log in
- Two integrity categories distinguish between production and development software and data:
  - Development (ID): development entities
  - Production (IP): production entities
- Security Categories:
  - Production (SP): production code and data
  - Development (SD): same as (previous) security category Development (D)
  - System Development (SSD): same as (previous) security category System Development (SD)

Assign Classes/Categories to Users

- The security clearances of all classes of users remain equivalent to those of the model without integrity levels and categories. The integrity classes are chosen to allow modification of data and programs as appropriate. For example, ordinary users should be able to modify production data, so users of that class must have write access to integrity category IP. The following listing shows the integrity classes and categories of the classes of users:

<table>
<thead>
<tr>
<th>Users</th>
<th>Security clearance</th>
<th>Integrity clearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordinary users</td>
<td>(SL, {SP})</td>
<td>(ISL, {IP})</td>
</tr>
<tr>
<td>Application developers</td>
<td>(SL, {SD})</td>
<td>(ISL, {ID})</td>
</tr>
<tr>
<td>System programmers</td>
<td>(SL, {SSD})</td>
<td>(ISL, {ID})</td>
</tr>
<tr>
<td>System controllers</td>
<td>(SL, {SP,SD})</td>
<td>(SP, {IP,IP})</td>
</tr>
<tr>
<td>System managers and auditors</td>
<td>(AM, {SP,SD,SSD})</td>
<td>(ISL, {IP,IP})</td>
</tr>
<tr>
<td>Repair</td>
<td>(SL, {SP})</td>
<td>(ISL, {IP})</td>
</tr>
</tbody>
</table>
Assign Classes/Categories to Objects

- The final step is to select integrity classes for objects. Consider the objects Production Code and Production Data. Ordinary users must be able to write the latter but not the former. By placing Production Data in integrity class (ISL, {IP}) and Production Code in class (IO, {IP}), an ordinary user cannot alter production code but can alter production data. Similar analysis leads to the following:

<table>
<thead>
<tr>
<th>Objects</th>
<th>Security level</th>
<th>Integrity level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development code/test data</td>
<td>(SL, {SD})</td>
<td>(ISL, {IP})</td>
</tr>
<tr>
<td>Production code</td>
<td>(SL, {SP})</td>
<td>(IO, {IP})</td>
</tr>
<tr>
<td>Production data</td>
<td>(SL, {SP})</td>
<td>(ISL, {IP})</td>
</tr>
<tr>
<td>Software tools</td>
<td>(SL, 0)</td>
<td>(IO, {ID})</td>
</tr>
<tr>
<td>System programs</td>
<td>(SL, 0)</td>
<td>(ISP, {IP, ID})</td>
</tr>
<tr>
<td>System programs in modification</td>
<td>(SL, {SSD,})</td>
<td>(ISL, {ID})</td>
</tr>
<tr>
<td>System and application logs</td>
<td>(AM, (appropriate categories))</td>
<td>(ISL, 0)</td>
</tr>
<tr>
<td>Repair</td>
<td>(SL, {SP})</td>
<td>(ISL, {IP})</td>
</tr>
</tbody>
</table>

Operation/Comparison of the Model

- The repair class of users has the same integrity and security clearance as that of production data, and so can read and write that data.
- It can also read production code (same security classification and (IO, {IP}) dom (ISL, {IP})), system (same as (ISL, {SP}) dom (SL, 0) and (ISP, {IP, ID}) dom (ISL, {IP})), and repair objects (same security classes and same integrity classes);
- it can write, but not read, the system and application logs (as (AM, {SP}) dom (SL, {SP}) and (ISL, {IP}) dom (ISL, 0)).
- It cannot access development code/test data (since the security categories are disjoint), system programs in modification (since the integrity categories are disjoint), or software tools (again, since the integrity categories are disjoint).
- Thus, the repair function works as needed.
- Lipner’s model demonstrates that the Bell-LaPadula Model can meet many commercial requirements, even though it was designed for a very different purpose. The resiliency of that model is part of its attractiveness; however, fundamentally, the Bell-LaPadula Model restricts the flow of information. Lipner notes this, suggesting that combining his model with Biba’s may be the most effective.
Clark-Wilson Integrity Model

- Use transaction as basic operation. More accurately model the commercial systems.
- CDI: Constrained data item. Data subject to integrity control.
- Two procedures:
  - Integrity verification procedure (IVP): test the CDIs conform to the integrity constraints at the time IVPs are run.
  - Transformation procedure (TP): change the state of the data in the system from one valid state to another.
- Two kinds of rules: Certification rules and Enforcement rules.

Certification Rules/Enforcement Rules

- **Certification rule 1 (CRI):** When any IVP is run, it must ensure that all CDIs are in a valid state.
- **Certification rule 2 (CR2):** For some associated set of CDIs, a TP must transform those CDIs in a valid state into a (possibly different) valid state.
- **Enforcement rule 1 (ER1):** The system must maintain the certified relations, and must ensure that only TPs certified to run on a CDI manipulate that CDI.
- **Enforcement rule 2 (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. If the user is not associated with a particular TP and CDI, then the TP cannot access that CDI on behalf of that user.

This defines a set of triples \((user, TP, \{CD1 set\})\) to capture the association of users, TPs, and CDIs. Call this relation allowed \(A\). Of course, these relations must be certified:
Additional Rules

- **Enforcement rule 3 (ER3):** The system must authenticate each user attempting to execute a TP.
- **Certification rule 4 (CR4):** All TPs must append enough information to reconstruct the operation to an append-only CDI.
- **Certification rule 5 (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.
- **Enforcement rule 4 (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.

Satisfy the Requirements

- Requirement 1. If users are not allowed to perform certifications of TPs, but instead only "trusted personnel" are, then CR5 and ER4 enforce this requirement. Because ordinary users cannot create certified TPs, they cannot write programs to access production databases. They must use existing TPs and CDIs—that is, production programs and production databases.
- Requirement 2. This requirement is largely procedural, because no set of technical controls can prevent a programmer from developing and testing programs on production systems. (The standard procedural control is to omit interpreters and compilers from production systems.) However, the notion of providing production data via a special process corresponds to using a TP to sanitize, or simply provide, production data to a test system.
Satisfy the Requirements

- Requirement 3. Installing a program from a development system onto a production system requires a TP to do the installation and "trusted personnel" to do the certification.
- Requirement 4. CR4 provides the auditing (logging) of program installation. ER3 authenticates the "trusted personnel" doing the installation. CR5 and ER4 control the installation procedure (the new program being a UDI before certification and a CDI, as well as a TP in the context of other rules, after certification).
- Requirement 5. Finally, because the log is simply a CDI, management and auditors can have access to the system logs through appropriate TPs. Similarly, they also have access to the system state.

Compared with Biba Model

- Recall that the Biba model attaches integrity levels to objects and subjects.
- In Clark-Wilson Model, each object has two levels: constrained or high (the CDIs) and unconstrained or low (the UDIs). Similarly, subjects have two levels: certified (the TPs) and uncertified (all other procedures).
- Clark-Wilson Model has certification rules. Biba doesn't.
- Clark-Wilson has procedure to verify trusted entities and their actions.
- Clark-Wilson requires a trusted entity certifies the method of upgrading integrity level. More practically than Biba which requires pass on every input of integrity level changes to a higher level entities.
Exercises

- Explain why Repair Users are associated with label (SL, \{ SP \}) (ISL, \{ IP \}) and Software Tools are associated with labels (SL, Ø) (IO, \{ ID \}).

  Ans: Repair Users need to read and execute Software Tool. To read the software tool, the security label of the Repair Users need to dominate the software tool. Here we have SL == SL, Ø \subseteq\{SP\}, therefore (SL, \{SP\}) dom (SL, Ø). However, the categories in the integrity labels, \{IP\} and \{ID\}, are disjoint. Therefore Repair Users can not use the Software Tools for development. It must use tools integrated with them for repairing software in the production environment.

- Software Tools are associated with labels (SL, Ø) (IO, \{ ID \}) and Application developers are associated with labels (SL, \{ SD \}) (ISL, \{ ID \}). Explain how can these label assignments facilitate the access control.

  Ans: The security label of Application Developers dominates that of the Software Tools. (SL, \{SD\}) dom (SL, Ø). The integrity label of Software Tools dominates that of Application Developers. (IO, \{ID\}) dom (ISL, \{ID\}). Therefore Application developer can read and execute the software Tools, but they cannot overwrite the Software Tools.

- Which group of users can move the development code to the production environment?
  Ans: System Controllers.