Access Models and Integrity

Commercial v Government Goals

- Control of confidentiality is paramount in government/military
- Control of integrity is paramount in commercial environment
- Can they use the same policy models?
- Are any new mechanisms required for integrity enforcement?

Integrity Issue

- Bell-LaPadula and Biba dual
  - But, Bell-LaPadula has some practical analogues
- Biba is impractical because
  - Higher integrity processes may legitimately use lower integrity data
  - There are so many instances that upgraders are impractical (because they are domain-specific)
- We examine integrity in more detail and discuss alternatives

Integrity Example: SSH

SSH Integrity

- Privileged operations
  - Key exchange and rekeying
  - Authentication
  - Create and cleanup pseudo-terminals
- Depends on integrity of
  - Users’ public key files
- Low integrity data
  - Read remote user inputs
  - Execute user programs (done as user identity in UNIX)

Integrity Management Options

- System Integrity Enforcement
  - Biba integrity
  - Low water mark (LOMAC)
  - Domain and type enforcement (DTE)
  - Clark-Wilson
- Managed integrity
- Language-based integrity
- Privilege separation
**Biba Integrity**

![Diagram of Biba Integrity](image)

- Dominance: $1 > 2$
- Biba Operations:
  - Read
  - Write

**Low Water Mark**

- Dynamic model over subject integrity labels
  - The lowest integrity label of the objects read by a subject

**References**


**Self-Revocation Problem in LOMAC**

- Self-Revocation Problem
  - Process starts with high integrity
  - Open high integrity object o1
  - Read from low integrity object o2
  - LOMAC semantics reduce process’s integrity to low
  - No longer can write to o1
  - Inconsistent with UNIX semantics for open files

**Self-Revocation Example**

- **Step 1**: initial state
  - ps
  - pipe
  - grep

- **Step 2**: ps reads low file
  - pipe

- **Step 3**: demotion
  - ps
  - pipe
  - pipe
  - grep
  - grep
  - ps

- **Step 4**: pipe write denied

**Unnamed Pipe Possession Rule**

- **Step 1**: initial state – unnamed pipes simply connect subjects of same level
  - ps
  - pipe

- **Step 2**: ps reads low file
  - pipe

- **Step 3**: demotion of all processes in process group and pipe
  - ps
  - pipe

- **Step 4**: pipe write succeeds
  - pipe

**Unnamed Pipe Usage Rule**

- **Previous does not work when**
  - Processes in different groups have access to same unnamed pipe
  - Occurs when a shell creates a new process group via `setpgrp`
    - Shell forks source and sink processes and gives them the unnamed pipe

- **Solution**:
  - Unnamed pipe is owned by first process group to use it
  - Works because the shell does not actually use the pipe
  - Slightly different approach used for shared memory
SSH and LOMAC

Dominance

\[ L_1 > L_2 \]

Requests write: Would lower sshd to L3

Remote user

\[ L_1 \]

\[ L_2 \]

\[ L_3 \]

Integrity Level Dependency

- What does a program’s integrity impact depend on?
  - User (Biba)
  - Job(s) (RBAC)
  - Objects read (LOMAC)
  - Objects modified
  - Objects executed
- Differences between Objects Read and Objects Executed
  - Objects Read is superset of Objects Executed
  - Objects Executed may control integrity of Objects Read
  - Thus, Objects Read may be too aggressive

Domain and Type Enforcement

- Type Enforcement with
  - Subject transitions based on the program being run (objects executed)
- Since Type Enforcement is as access matrix model
  - Mapping from objects to integrity level is implicit
  - Mapping assumes any low integrity object reads can be sanitized
  - Mapping could be wrong

Type Enforcement [BoebertKain84]

- Assume that sshd handles it

SSH and DTE

Subject Types:
- sshd
- remote_user

Requests write: Assume that sshd handles it

Remote user

\[ L_1 \]

\[ L_2 \]

SAFKASI – Language-based integrity

- Run untrusted code in same address space as trusted code
  - Assumes type safety of language
- Use special operation (e.g., BeginPrivilege()) to define privileged operation
- Prevent the “Confused Deputy Problem”
  - Code is invoked in an environment controlled by the attacker
  - SELinux problem – specify faulty input library and log location to overwrite password file
Clark-Wilson Integrity

- **Goal**
  - No user, even if authorized, may be permitted to modify data items in such a way that assets are lost or corrupted.
- **Pre-computer commercial policy goals**
  - Separation of duty
  - Well-formed transactions
- **Model Intuition**
  - Base secure computing on well-formed transactions
  - Provide independent testing (e.g., balancing the books)
  - Enable enforcement of separation of duty

Well-formed Transaction

- **Definition**
  - Prescribed means of modifying data
  - With means to preserve data integrity
- **Example: Double Entry Bookkeeping**
  - Any modification must consist of two parts
    - Action and a balance
  - E.g., When writing a check there must a corresponding entry in accounts payable

Separation of Duty

- **Definition**
  - Divide a process into subparts
  - Each subpart is performed by a different person
- **Example**
  - A person who certifies a well-formed transaction may not execute it
- **Susceptible to collusion**
- **Static vs Dynamic Separation of Duty**
  - Static: Each user has a specific subpart
  - Dynamic: Can perform a subpart, but is then excluded from another
- Typically, execution subtasks are required to be separated from certification subtasks

Mandatory Commercial Control

- **Authenticate users (same as for military)**
- **Associate permissions with programs that implement well-formed transactions**
- **Each user is associated with a valid set of programs to be run**
- **Audit relevant events (similar to accountability in military, but different events)**
- Both require a reference monitor to enforce policies

Clark-Wilson Model

- **Object labels:** Constrained and unconstrained data items
  - Apply integrity enforcement to constrained data items
- **Processes:** Transformation Procedures
  - Well-formed transaction
- **Auditing:** Integrity Verification Procedures
  - Verify that CDIs conform to integrity specification
- **Semantics**
  - IVPs prove system is in a valid state
  - Only TPs can modify CDIs
  - CDIs modified by other processes are effectively downgraded
  - Repeat

Certification and Enforcement

- **Summary – Data-TP assignment**
  - Certification (C1): IVPs ensure CDIs are valid
  - Certification (C2): TPs are valid
    - Take CDIs from one valid state to another
    - TP-CDI relation: sets of CDIs for a TP
  - Enforcement (E1): Only manipulation of a CDI is by a TP as specified in C2
- **Separation of Duty**
  - Enforcement (E2): TP-user-CDI – not solely TP-user
  - Certification (C3): E2 relations must be certified for SoD
Certification and Enforcement (con’t)

- **Authentication**
  - Enforcement (E3): Must authenticate users who execute TPs
  - Why no certification of this mechanism?
- **Audit**
  - Certification (C4): All TPs are certified to write append-only logs of security decisions
- **Handling of low-integrity data**
  - Certification (C5): TPs that depend on UDIs must be certified to perform valid UDI transforms (to CDI) or rejects
  - Enforcement (E4): Only authorized agents can certify
  - E4 makes policy mandatory

SSH and Clark-Wilson

Clark-Wilson Requires:
- C1: IVP to verify inputs
- C2: CDIs for sshd
- E1: CDIs mod by TPs
- E2: Admin is user
- C3: Certify sshd users
- E3: sshd authenticates
- C4: sshd logging
- C5: sshd handles UDIs
- E4: valid certification

SSH and Managed Integrity

Managed Integrity

- For an integrity target (e.g., DTE integrity)
- Find integrity conflicts
  - Policy statements that violate conflict
- Resolve conflicts
  - Change system: Remove subjects or objects
  - Clark-Wilson: Specify means for handling low integrity data
  - LOMAC: Artifacts that enable resolution
  - Language: Programming language primitives to enable management
  - Policy: Modify constraints or permissions

Application Decomposition

- Separate application into privileged and unprivileged components
  - Remote input to unprivileged instead
  - Do as much processing in unprivileged as possible
  - For few remaining requests, unprivileged sends formatted input to privileged
  - Complete service’s function
  - Have to transfer state from sshd-unprivileged to user’s sshd via privileged
SSH and Privilege Separation

Requests write: Filtered requests only

Unprivileged/Unprivileged ssh

Requests write: To unprivileged first

Remote user

Access Control Models

- Aggregated Models
  - Useful when aggregations are fixed and have a practical analogue
  - Inheritance and constraints are not well understood
  - Constraints are needed for safety (do we meet security goals)
- Lattice Access Control Models
  - Derived from military secrecy (have a practical analogue)
  - Safety is implicit, but downgraders are ad hoc
  - Can verify lattice properties statically when labels are static
- Domain Transitions and Integrity
  - Dynamic subject related triggered by execution or read
  - Low integrity data must be handled by higher integrity processes
  - Solutions: lower subjects, implicit integrity, certification, explicitly manage, privilege separation
- Other Models
  - Predicate Models: dynamic rule-based approach
  - Safety Models: Limited models to enable verifiable safety

Clark-Wilson Mechanisms Needed

- Clark-Wilson Requirements
  - Ensure a data item are manipulated only by well-formed transactions
  - Verify integrity of well-formed transaction code
  - Protect integrity of well-formed transaction code
  - Limit user to subset of programs based on separation of duty
- A very different mechanism is needed
  - Justified by Multics having distinct mechanisms for MLS and DAC
  - However, SELinux uses the same mechanism for MLS and DTE (slightly different policy generation)