Politecnico di Milano
Dip. Elettronica e Informazione
Milano, Italy

Authorization and access control

Access control: deceivingly simple

- A binary decision:
  - Access either allowed or denied
- However:
  - How do we *design* the access rules?
  - How do we *express* the access rules?
  - How do we *store* them
  - How do we appropriately *apply* them?
- Access control policies
- Access control models
  - Design and expression of policies; storage (in part)
  - A model should be *complete* and *consistent*
- Access control mechanisms and implementation
  - Storage (in part), application at runtime

Access control policies

- They can be roughly divided in
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
- The key difference between *discretionary* and *mandatory* access control is that in discretionary system the *owner* of a resource can assign privileges on it, including ownership, to others. In mandatory systems, only a security administrator can set privileges
- All of the common COTS operating systems are DAC based
  - MAC extensions are available for Linux, BSD, and other systems

Examples of well-known DAC systems

- Unix
  - Subjects: users, groups
  - Objects: files
  - Actions: read, write, execute
- Windows (in the NT family, inc. business versions of XP and Vista)
  - Subjects and objects as above but with “roles” instead of groups, multiple ownership of users and roles over files
  - Added actions: delete, change permissions, change ownership

Modeling access controls in DAC

- We need to model the following entities:
  - **Subjects** who can exercise privileges
  - **Objects** on which privileges are exercised and
  - **Actions** which can be exercised
- A **protection state** is commonly defined (Lampson 71, Graham and Denning 72, Harrison, Ruzzo and Ullmann 76) by a triple (S, O, A) where A is represented by a **matrix** with S on rows and O on columns. A[s,o] represents the privileges of subject s over object o

	Alpha	*Beta*	*Gamma*
Alice	Read	Write	
Bob		Read/Write	
Charlie	Write		Read/Write

States and transitions in the HRU model

- Basic operations
  - create subject <s>; create object <o>; enter <permission> into [s,o]; delete r from [s,o]; destroy subject s; destroy object o
- Transitions are combinations of commands
  - E.g. create file (subject u; file f) means: create object f; enter "own" into [u,f]; enter "read" into [u,f]; enter "write" into [u,f]
  - Is this right? No, we need to check if f existed before, otherwise u would be stealing it away!
  - We need an "if" construct into transitions
- Enter the safety problem

Safety problem

- Given a set of transitions and an initial configuration, can a certain subject attain a certain right on a certain object?
  - Obviously, yes: if the owner allows it!
  - But, if the owner does not allow it or does not exist?
  - If it happens the set of commands is *unsafe by design !*
- More formally, given a configuration Q, is there any sequence of transitions which leaks a certain right r (for which the owner is removed) into the access matrix somewhere?
  - If not, then the system is safe wrt right r
  - In a general HRU model **this is undecidable**
  - Decidable only in mono-operational systems, which are substantially useless (e.g. you cannot create a file and own it), or if subjects/objects are finite

Take-grant model

- Restricts the model to be able to prove safety
- We model privileges as a directed **graph** where S and O are nodes, and privileges are edges:
  - Read, write
  - Take: a node with take privilege wrt another can take (not in the sense of remove) any privilege from the latter
  - Grant: a node with grant privilege wrt another can give it any privilege it owns
- Operations are take, grant, create (a new object with certain privileges) and remove (certain privileges from an object)
- Safety here is decidable and O(n) wrt a specific privilege, and O(n^3) globally
- A pity this is substantially useless in real world...

Common DAC implementations

- DAC implementations are substantially reproduction of HRU models
- Access matrix is a sparse matrix = lots of wasted space
- Alternative implementations:
  - **Authorizations table**: records non-null triples S-O-A, tipically used in DBMS
  - **Access Control Lists**: records by column (i.e. for each object, the list of subjects and authorizations)
  - **Capabilities Lists**: records by row (i.e. for each subject, the list of objects and authorizations)
- ACLs and capabilities obviously have different pros and cons

ACL vs capability lists

- ACL needs *subjects* to be securely authenticated
- ACL is more efficient if privilege assignments and revocations happen *per object*
- This is what usually happens, and this is why most systems (POSIX, Win) use ACLs
- Some systems (e.g. POSIX) use abbreviated ACLs

- Capability needs complex and verified methods for propagation and appropriate identification of objects
- Capabilities more efficient if privilege assignments and revocations happen *per subject*
- Capabilities are optional in POSIX (Linux and BSD)

Generic DAC shortcomings

- Cannot prove safety
- Control access to objects but not to information inside objects
  - E.g. susceptible to *trojan horse* problems even if users are trustworthy: if a program executes malicious actions, it will do so with the privileges of the user, and will be able to tamper with any file owned by the user
- Problems of scalability and management, since each user can potentially compromise security of the system with his/her own decisions

Mandatory access control policies

- Generic approach: define a *classification* of subjects (clearance) and objects (sensitivity)
- A classification is generally a partial order relationship, with a dominance concept which we will denote as ≥
- Often the classification is composed of
  - A strictly ordered **secrecy level** (e.g. the US classification: Unclassified < FOUO < Secret < Top Secret; the NATO levels: Unclassified < NATO Confidential < COSMIC Secret < COSMIC Top Secret)
  - A set of labels (e.g. “Policy”, “Energy”, “Finance”; or ATOMAL)
- Dominance is usually defined as
  - $\{C1,L1\} \geq \{C2,L2\} \Leftrightarrow C1 \geq C2$ and $C2 \subseteq C1$

Just to get all formal …

- This relationship is a lattice:
  - Reflexive
  - Transitive
  - Anti-symmetric
- For each couple of objects I can identify a least upper bound (an object which dominates both, and which in turn is dominated by any other object dominating both) and a greatest lower bound (an object which is dominated by both, and which in turn dominates any other object which is dominated by both)

Secrecy constraints; Bell-LaPadula model

- Subject s can read object o iff s dominates o (no read up property, aka simple security property)
- Subject s can write object o iff o dominates s (no write down property, aka *-property)
  - For this reason, a user may be allowed to connect with any security level which he currently dominates
- Tranquillity property: security classifications cannot be changed (clearance can)
- This would create a monotonic flow of information towards higher classification

Limitations of a BLP MAC model

- Sanitization: the output of a process can be of lower sensitivity than the inputs
- Declassification: at times, information is not secret anymore (think of WW2 battle plans today)
- The *association* of two values may have a higher sensitivity than each
- The *aggregation* of some data may be more sensitive than each data point singularly
- *Covert channels* may still be an issue
  - e.g. a TS process which cannot be directly observed by a S user can be identified by the response time of the CPU
  - System design must take care of noninterference between processes at different security levels

Integrity constraints: BIBA

- Dual of BLP for integrity
  - Instead of secrecy level, integrity level
  - User integrity: how much trust we have in user and correctness of his information
  - Object integrity: how much value the correctness of the information has, and how much it is trusted
- Dual rules
  - Subject s can write object o iff s dominates o (no write up property aka *-property)
  - Subject s can read object o iff o dominates s (no read down aka simple security property)
- Biba and BLP can coexist but with **different** sets of classifications !

Relational multilevel systems

- Multilevel security can be applied also to RDBMS
- Each tuple, or even each attribute of a tuple, can have an associated security classification
- Each user will see a view comprising only those tuples and attribute (s)he dominates
- Each user will write at his/her own security level
- This leads to the problem of poli-instantiation, because a user at lower level can instantiate a tuple with the same primary key of an existing tuple which is classified at a level he cannot access

Access control mechanisms

- Access control usually based on a *reference monitor*
  - Tamper proof
  - No bypass
  - Small enough to be verifiable
  - In other words, a *security kernel*
- A number of potential vulnerabilities
  - Storage channels: re-assigned resources such as RAM should be cleared befor reassignment
  - Side channels: sometimes timing, or other issues, can reveal hidden information