Authorisation and Access Control

Daniel Bosk

Department of Information and Communication Systems,
Mid Sweden University, SE-851 70 Sundsvall.

23rd April 2017

- A policy specifies who is allowed to do what.
- Access control enforces operational security policies.

Definition

- We have an active entity: a *subject* (representing a *principal*).
- The subject tries to access an *object* with some *access operation*.
- To protect this, there is a *reference monitor* granting or denying this access.

Definition (Authentication)

- Principals make statements.
- Let s be a statement.
- Authentication answers ‘Who said s?’ by stating a principal.

Definition (Authorization)

- Let o be an object.
- Authorization answers ‘Who is trusted to access o?’ by stating a (list of) principal(s).

Definition (Authentication)

- Principals make statements.
- Let s be a statement.
- Authentication answers 'Who said s?' by stating a principal.

Definition (Authorization)

- Let o be an object.
- Authorization answers 'Who is trusted to access o?' by stating a (list of) principal(s).

Idea: Reference monitor

- The reference monitor requires authentication of principals to be able to authorize the subject it represents.
- By authorization the reference monitor decides whether to grant or deny a subjects request for access to an object.
- For this decision the reference monitor must use the security policy.

Definition

- The elementary access modes for operations are to *observe* or to *alter* a resource.
- Different *access operations* requires combinations of access modes.

Definition

- An *access right* is a right to perform an access operation.
- *Privileges* are sets of access rights.

Definition

- The elementary access modes for operations are to *observe* or to *alter* a resource.
- Different *access operations* requires combinations of access modes.

Definition

- An *access right* is a right to perform an access operation.
- *Privileges* are sets of access rights.

Example (BLP)

- The Bell-LaPadula (BLP) model has four access rights:
  - Execute
  - Read
  - Append (blind write)
  - Write
- These rights requires the two different modes:

Execute	requires none.
Append	requires alter.
Read	requires observe.
Write	requires observe and alter.

Example (Reference monitor)

- In a multi-user OS, processes uses the open(2) system call to request access.
- The OS makes sure no conflicting accesses are granted.
- Note that some things can be used without a direct request.
- E.g. the user doesn't need read permission to execute a program.

Example (UNIX-like systems)

- In UNIX-like systems we have three access operations:
  - Read
  - Write
  - Execute
- These are applied to both files and directories, but differently for each.
- You can read from a file, or list the content of a directory.
- You can write contents to a file, or create or rename files in a directory.
- You can execute the file, or you can search the directory.
- Operations on subdirectories or files are thus handled by access operations to the parent directory.

Example

Policies for creating and deleting files are expressed by

- access control on the directory in UNIX-like systems, but
- specific create and delete right in Windows.

Example

Policies for defining security settings such as access rights are handled by

- access control on the directory in UNIX-like systems, but
- could be handled by right like grant and revoke.

Example

Policies for creating and deleting files are expressed by

- access control on the directory in UNIX-like systems, but
- specific create and delete right in Windows.

Example

Policies for defining security settings such as access rights are handled by

- access control on the directory in UNIX-like systems, but
- could be handled by right like grant and revoke.

- We can adapt two different focuses on the policy.
- The first being, "What is a principal allowed to do?"
- The second, "What may be done with an object?"
- Which one is suitable depends on the application.
- E.g. an OS usually takes the second approach as its purpose is to manage objects.
- E.g. applications like databases might focus on what different users are allowed to do.

- We can adapt two different focuses on the policy.
- The first being, "What is a principal allowed to do?"
- The second, "What may be done with an object?"
- Which one is suitable depends on the application.
- E.g. an OS usually takes the second approach as its purpose is to manage objects.
- E.g. applications like databases might focus on what different users are allowed to do.

- The access control structure is used to store an implemented policy.
- This structure should help to express the policy.
- Access rights for each combination of subject and object should be possible to define.
- The importance of the choice of structure is shown when the system scales up.

Definition (Access control matrix)

- S be the set of subjects,
- O the set of objects, and
- A the set of access operations.
- *Access control matrix*: $M = (M_{so})$, where $s \in S$ and $o \in O$.
- Each entry $M_{so} \subseteq A$ specifies the operations subject s may perform on the object o.

Definition (Access control matrix)

- S be the set of subjects,
- O the set of objects, and
- A the set of access operations.
- *Access control matrix*: $M = (M_{so})$, where $s \in S$ and $o \in O$.
- Each entry $M_{so} \subseteq A$ specifies the operations subject s may perform on the object o.

Note

- The access control matrix is an abstract concept.
- It's not very suitable for implementation.

- Capabilities focuses on the subject.
- Access rights are stored with the subject.
- Capabilities are essentially the rows of the access control matrix.
- Subjects may grant rights to other subjects.
- Maybe even grant right to grant rights.
- How do you know who may access what?
- How do you revoka a capability?

- Focuses on the objects.
- Access rights are stored with the object.
- ACLs are essentially the columns of the access control matrix.
- How do you check access right of a specified subject?

- Who sets the policies?
- The policy is the governing rules of who may access what.
- Who sets or is allowed to change the policy could be assigned to
  - the owner of a resource. This is called *discretionary* access control.
  - a system wide policy decreeing who is allowed access or not. This is called *mandatory* access control.

- To more easily manage access control for many subjects and objects we need another approach than above.
- The solution is to introduce intermediate levels of complexity.

- We might be able to use identity based acces control (IBAC).
- IBAC doesn't scale well.
- Thus we add groups to handle multiple principals at the same time, e.g. a computer security class.
- This makes things easier.

- Another approach is to use roles.
- A role is a collection of procedures assigned to users.
- At a first look it reminds a lot about groups.
- However, this is a more high-level way of handling access control.

- The procedures have more complex semantics than just read or write.
- They can only be applied to objects of given data types.
- E.g. transfering funds in a bank.
- RBAC is typically found at the application level.

- We can further have role hierarchies, i.e. relationships between roles.
- E.g. we can have a teacher and a teaching assistant role, where the teacher has all rights of the TA.
- Separation of duties is an important principle in security, i.e. when the same subject isn't allowed to do two related operations.
- There can be static and dynamic policies for separation of duties.

Flat RBAC Users are assigned to roles, permissions are assigned to roles. Hence users get permissions via roles.

Hierarchical RBAC Adds support for role hierarchies.

Constrained RBAC Adds separation of duties.

Definition

Policy enforcement point Inspects request and generates authorization request for PDP.

Policy decision point, PDP Evaluates requests against policies. Returns permit or deny.

Policy information point Can be used by PDP to access attribute databases.

Example (Attributes)

- Subject attributes, e.g. age, clearance, department, role, ...
- Action attributes, e.g. read, delete, write, ...
- Object attributes, e.g. type, owner, classification, location, ...
- Contextual attributes, e.g. time, location, ...

- Multics introduced protection rings.
- Low-level version of the high-level BLP.
- These are mainly implemented in hardware and used to protect integrity.
- Access control is based on which rings the subject and object are in.
- E.g. 0 contains kernel, 1 contains OS functionality, 2 contains utilities, and 3 is for user processes.

- Some resources in e.g. the University's Computer Science Department can be accessed by all students, other only by students in a particular class etc.
- Department creates groups "All" and "DT116G", "DT145G" and "DV026G".
- The groups DT116G and All are of course related, DT116G is a subgroup of All and should access everything All can access too.
- However, there is no such relation between DT116G and DT145G.

- We can use these comparisons for security policy decisions.
- Is the group of the subject requesting access a subgroup of the group allowed access?
- These relationships have a corresponding mathematical construction called partial ordering.

Definition

A *partial ordering* $\leq$ on a set L is a relation on $L \times L$ that is

- reflexive, $\forall a \in L, a \leq a$,
- transitive, $\forall a, b, c \in L, a \leq b \wedge b \leq c \implies a \leq c$,
- antisymmetric, $\forall a, b \in L, a \leq b \wedge b \leq a \implies a = b$.

If $a \leq b$, we say that a dominates b.

Definition

A *lattice* $(L, \leq)$ is a set L with a partial ordering $\leq$ such that for every two elements $a, b \in L$ there exists

- an least upper bound $u \in L$: $a \leq u, b \leq u$ and for all $v \in L$: $(a \leq v \wedge b \leq v) \implies u \leq v$.
- a greatest lower bound $l \in L$: $l \leq a, l \leq b$ and for all $k \in L$: $(k \leq a \wedge b \leq b) \implies k \leq l$.

Lattices of Security Levels