KULeuven DistriNet Research Group

Access control

Frank Piessens

(Frank.Piessens@cs.kuleuven.be)

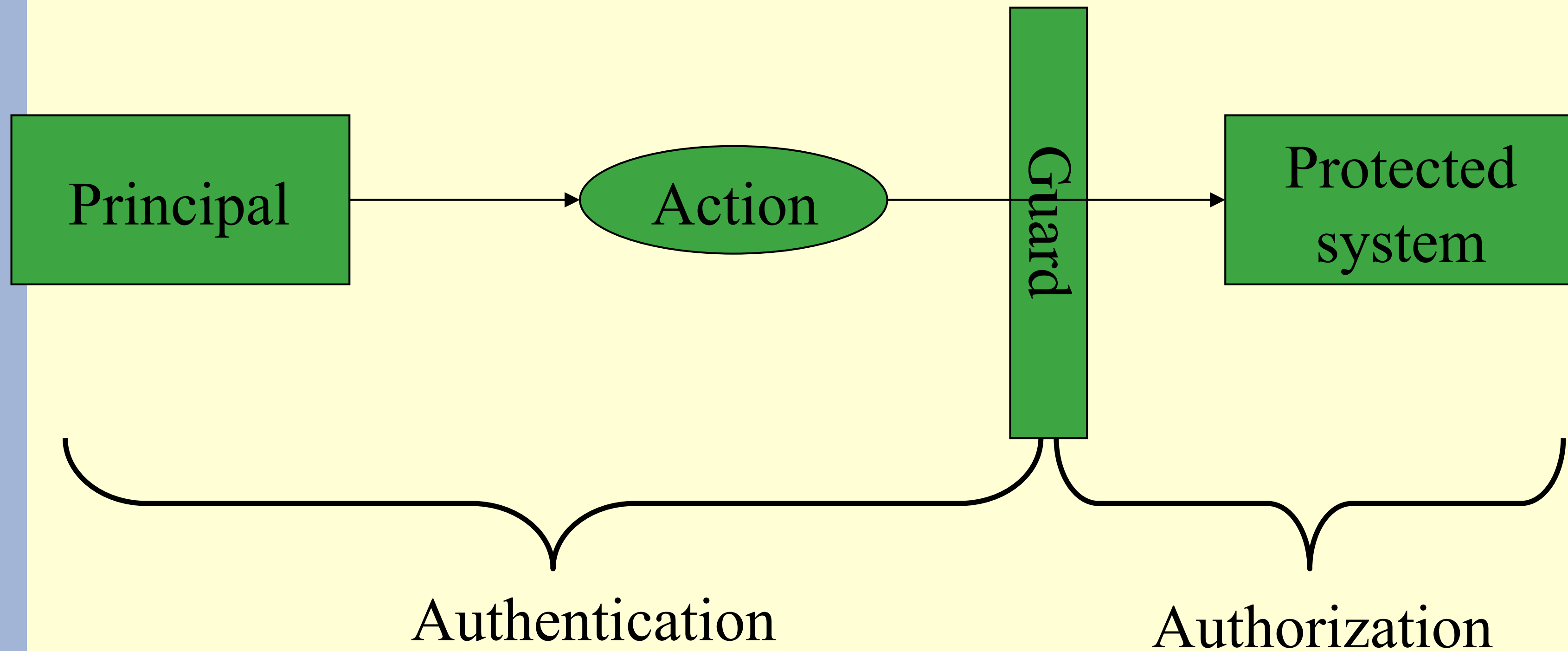
Overview

- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Access Control: introduction

- Security = prevention and detection of unauthorized actions on information
- Two important cases:
  - An attacker has access to the raw bits representing the information
    => need for cryptographic techniques
  - There is a software layer between the attacker and the information
    => access control techniques

General access control model



Examples

Principal	Action	Guard	Protected system
Host	Packet send	Firewall	intranet
User	Open file	OS kernel	File system
Java Program	Open file	Java Security Manager	File
User	Query	DBMS	Database
User	Get page	Web server	Web site
…	…	…	…

Entity Authentication

- Definition
  - Verifying the claimed identity of an entity (usually called *principal*) that the guard is interacting with
- Different cases need different solutions:
  - Principal is a (human) user
  - Principal is a (remote) computer
  - Principal is a user working at a remote computer
  - Principal is a user running a specific piece of code
  - …
- See separate session on entity authentication

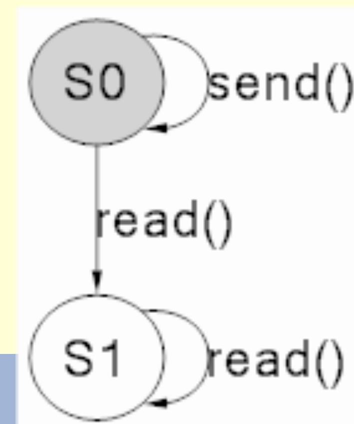
Authorization by the Guard

- Guard can have local state
  - "protection state"
- Upon receipt of an action
  - Decides what to do with the action
    - We only consider pass/drop
    - Alternatives are: modify/replace, first insert other action,…
  - If necessary: updates the local state
- Modeled by means of a "security automaton"
  - Set of states described by a number of typed state variables
  - Transition relation described by predicates on the action and the local state

Guard

- Notation:
  - Actions are written as procedure invocations
  - Behavior of the guard is specified by:
    - Declaration of state variables
      - Determine the state space
    - Implementations of the action procedures
      - Preconditions determine acceptability of action
      - Implementation body determines state update
- Example: no network send after file read

```
bool hasRead = false;
void send() requires  !hasRead {
  }
void read() {
  hasRead = true;
}
```



Policies and models

- Access control *policy* = rules that say what is allowed and what not
  - Semantics of a policy is a security automaton in a particular state
- Access control *model* = "A class of policies with similar characteristics"
  - Hard to define precisely
  - An access control model makes particular choices about what is in the protection state and how actions are treated

Overview

- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Discretionary Access Control (DAC)

- Objective = creator-controlled sharing of information
- Key Concepts
  - Principals are users
  - Protected system manages *objects*, passive entities requiring controlled access
  - Objects are accessed by means of *operations* on them
  - Every object has an *owner*
  - Owner can grant right to use operations to other users
- Variants:
  - Possible to pass on ownership or not?
  - Possible to delegate right to grant access or not?
  - Constraints on revocation of rights.

Security automaton for DAC

```
type Right = <User, Obj, {read, write}>;
Set<User> users = new Set();
Set<Obj> objects = new Set();
Set<Right> rights = new Set();  // represents the Access Control Matrix
Map<Obj,User> ownerOf = new Map();

// Access checks
void read(User u, Obj o) requires <u,o, read> in rights; {}
void write(User u, Obj o) requires <u,o,write> in rights; {}

// Actions that impact the protection state
void addRight(User u, Right <u',o,r>)
  requires (u in users) && (u' in users) && (o in objects) && ownerOf[o] == u; {
    rights[r] = true;
}
void deleteRight(User u, Right <u',o,r>)
  requires (u in users) && (u' in users) && (o in objects) && ownerOf[o] == u; {
    rights[r] = true;
}
```

Security automaton for DAC (ctd)

```
void addObject(User u, Obj o)
  requires (u in users) &&  (o notin objects); {
    objects[o] = true;
    ownerOf[o] = u;
}
void delObject(User u, Obj o)
  requires (o in objects) && (ownerOf[o] == u); {
    objects[o] = false;
    ownerOf[o] = none;
    rights = rights \ { <u',o',r'> in rights where o'==o};
}

// Administrative functions
void addUser(User u, User u') requires u' notin users; {
   users[u'] = true;
}
```

DAC

- Disadvantages:
  - Cumbersome administration
    - E.g user leaving the company or user being promoted to another function in the company
  - Not so secure:
    - Social engineering
    - Trojan horse problem

DAC Extensions

- Structuring users:
  - Groups
  - Negative permissions
  - But: insufficient to make administration much easier
- Structuring operations:
  - "access modes": observe / alter / …
  - Procedures: business procedure involving many operations on many objects
- Structuring objects:
  - E.g. Inheritance of folder permissions

Implementation structures

- DAC is typically not implemented with a centralized protection state
- Typical implementation structures include:
  - Access Control List: e.g. ACL's in Windows 2000
  - Capabilities: e.g. Open file handles in Unix
  - ...

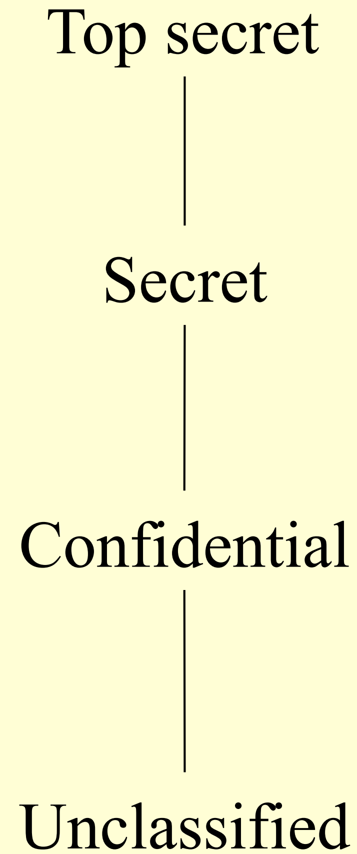
Overview

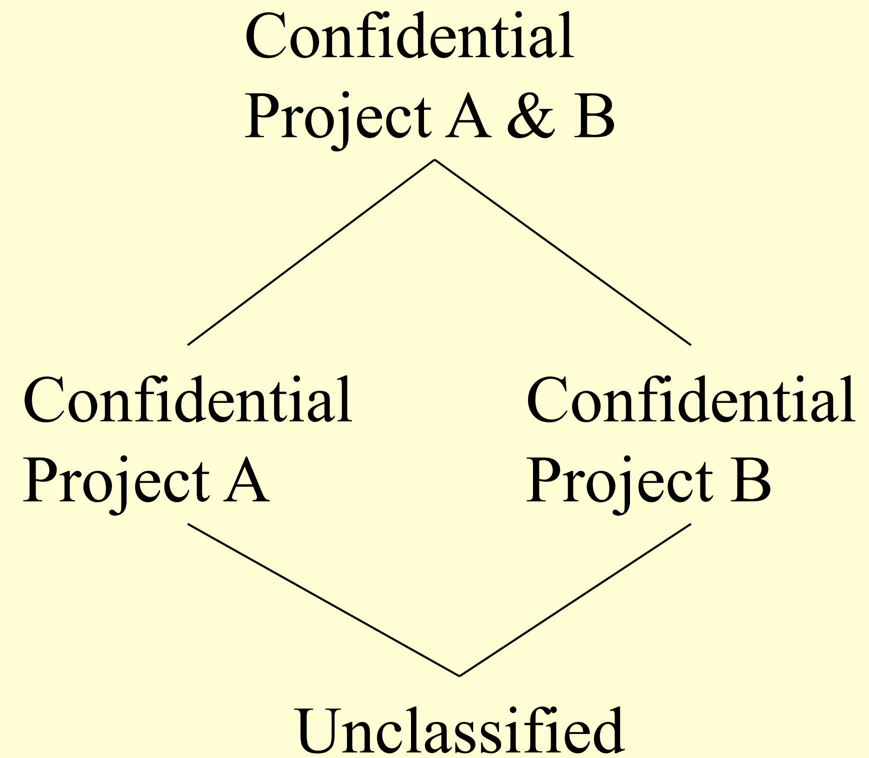
- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Mandatory Access Control (MAC)

- Objective = strict control of information flow
- Concrete example MAC model: Lattice Based Access Control (LBAC)
- Objective =
  - A lattice of *security labels* is given
  - Objects and users are tagged with security labels
  - Enforce that:
    - Users can only see information below their clearance
    - Information can only flow upward, even in the presence of Trojan Horses

Example lattices

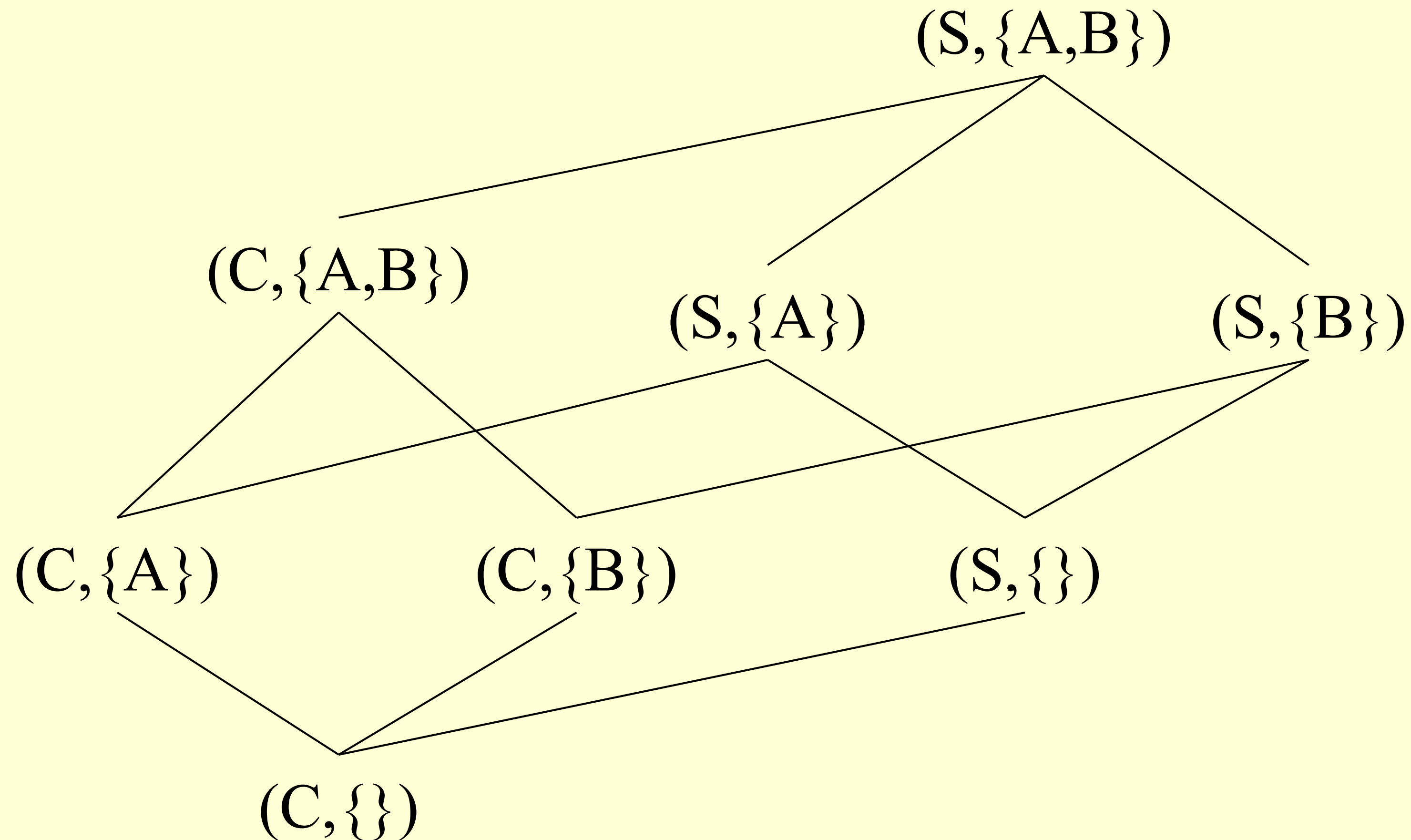




Typical construction of lattice

- Security label = (level, compartment)
- Compartment = set of categories
- Category = keyword relating to a project or area of interest
- Levels are ordered linearly
  - E.g. Top Secret – Secret – Confidential – Unclassified
- Compartments are ordered by subset inclusion

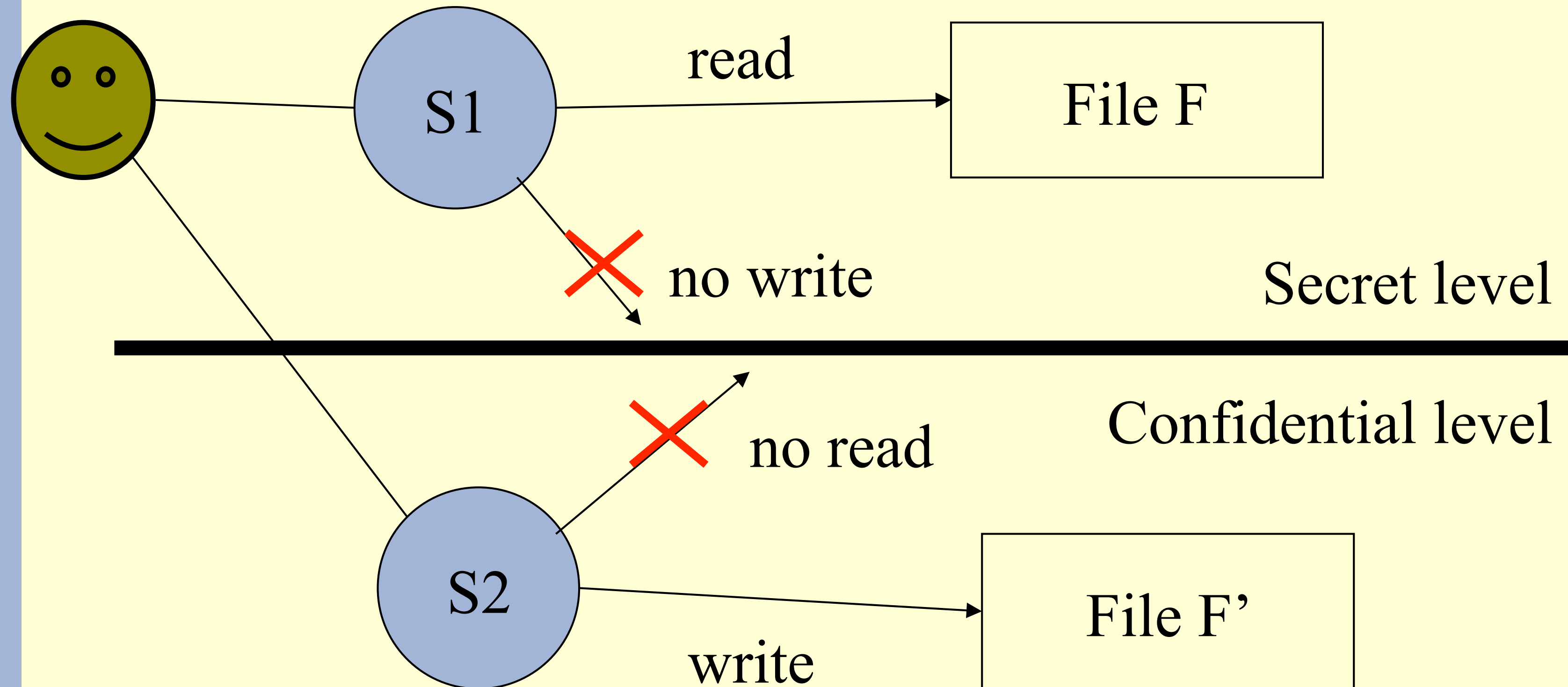
Example lattice



LBAC

- Key concepts of the model:
  - Users initiate *subjects* or *sessions*, and these are labeled on creation
  - Users of clearance L can start subjects with any label L' ≤ L
  - Enforced rules:
    - Simple security property: subjects with label L can only read objects with label L' ≤ L (no read up)
    - *-property: subjects with label L can only write objects with label L' ≥ L (no write down)
  - The *-property addresses the Trojan Horse problem

LBAC and the Trojan Horse problem



Security automaton for LBAC

```
// Stable part of the protection state
Set<User> users;
Map<User,Label> ulabel; // label of users

//Dynamic part of the protection state
Set<Obj> objects = new Set();
Set<Session> sessions = new Set();
Map<Session, Label> slabel = new Map(); // label of sessions
Map<Obj,Label> olabel = new Map(); // label of objects

// No read up
void read(Session s, Obj o)
  requires s in sessions && o in objects && slabel[s] >= olabel[o]; {}

// No write down
void write(Session s, Obj o)
  requires s in sessions && o in objects && slabel[s] <= olabel[o]; {}
```

Security automaton for LBAC (ctd)

```
// Managing sessions and objects
void createSession(User u, Label l)
  requires (u in users) && ulabel[u] >= l ; {
    s = new Session();
    sessions[s] = true;
    slabel[s] = l;
}

void addObject(Session s, Obj o, Label l)
  requires (s in sessions) &&  (o notin objects) && slabel[s] <= l; {
  objects[o] = true;
  olabel[o] = l;
}
```

LBAC

- Problems and disadvantages
  - Too rigid => need for “trusted subjects”
  - Not well suited for commercial environments
  - Covert channel problems

Overview

- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Role-Based Access Control (RBAC)

- Main objective: manageable access control
- Key concepts of the model:
  - Role:
    - many-to-many relation between users and permissions
    - Corresponds to a well-defined job or responsibility
    - Think of it as a named set of permissions that can be assigned to users
  - When a user starts a session, he can activate some or all of his roles
  - A session has all the permissions associated with the activated roles

Security automaton for RBAC

```
// stable part of the protection state
Set<User> users;
Set<Role> roles;
Set<Permission> perms;
Map<User, Set<Role>> ua; // set of roles assigned to each user
Map<Role, Set<Permission>> pa; // permissions assigned to each role

// dynamic part of the protection state
Set<Session> sessions;
Map<Session,Set<Role>> session_roles;
Map<User,Set<Session>> user_sessions;

// access check
void checkAccess(Session s, Permission p)
  requires s in sessions && Exists{ r in session_roles[s]; p in pa[r]}; {
}
```

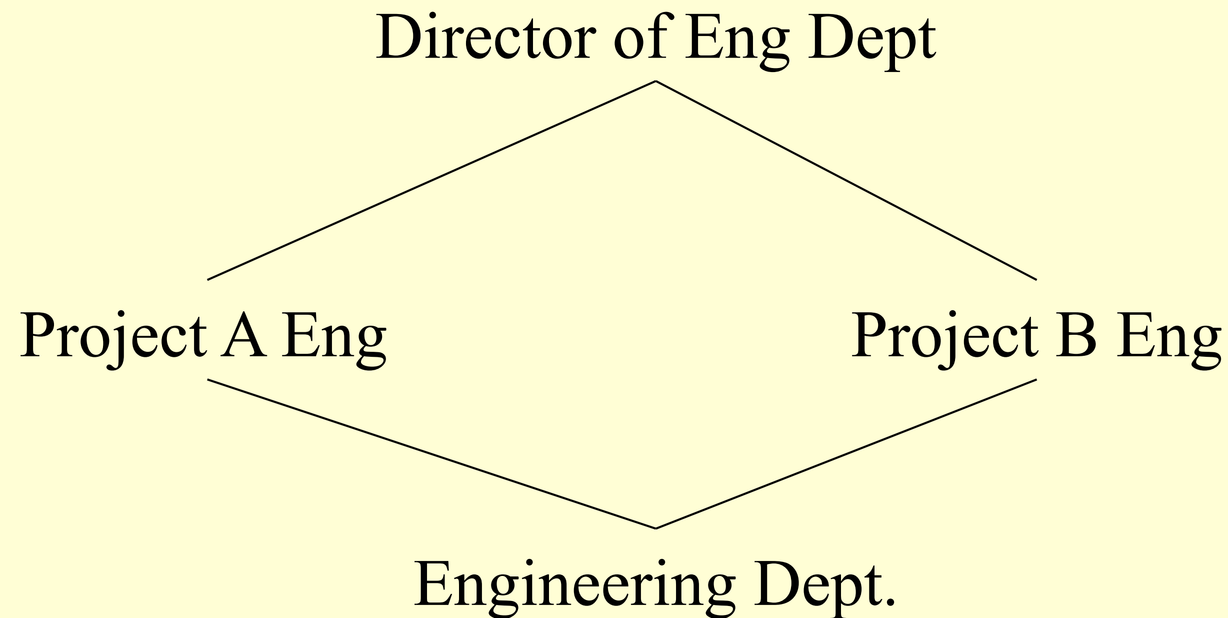
Security automaton for RBAC (ctd)

```
void createSession(User u, Set<Role> rs)
  requires (u in users) && rs < ua[u]; {
    Session s = new Session();
    sessions[s] = true;
    session_roles[s] = rs;
    user_sessions[u][s] = true;
}

void dropRole(User u, Session s, Role r)
  requires (u in users) && (s in user_sessions[u])
             && (r in session_roles[s]); {
  session_roles[s][r] = false;
}
```

RBAC - Extensions

- Hierarchical roles: senior role inherits all permissions from junior role



RBAC - Extensions

- Constraints:
  - Static constraints
    - Constraints on the assignment of users to roles
    - E.g. Static separation of duty: nobody can both:
      - Order goods
      - Approve payment
  - Dynamic constraints
    - Constraints on the simultaneous activation of roles
    - E.g. to enforce least privilege

RBAC in practice

- Implemented in databases or into specific applications
- Can be “simulated” in operating systems using the group concept
- Implemented in a generic way in application servers

Overview

- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Other Access Control Models

- Biba model: enforcing integrity by information flow
- Chinese wall model: dynamic access control model
  - "A consultant can only see company confidential information of one company in each potential-conflict-of-interest class"
- Theoretical models to study theoretical limits of security decision problems
- …

Overview

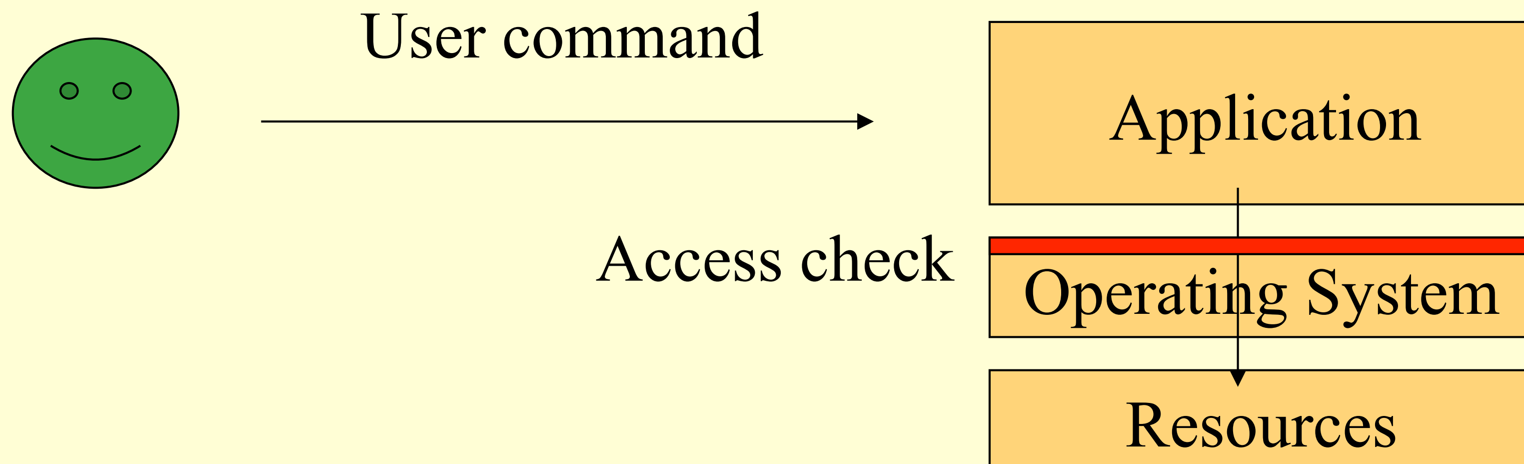
- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Implementing Access Control in Applications

- Basically three options
  1. Delegate to OS
  2. Rely on middleware / application server
  3. Roll your own

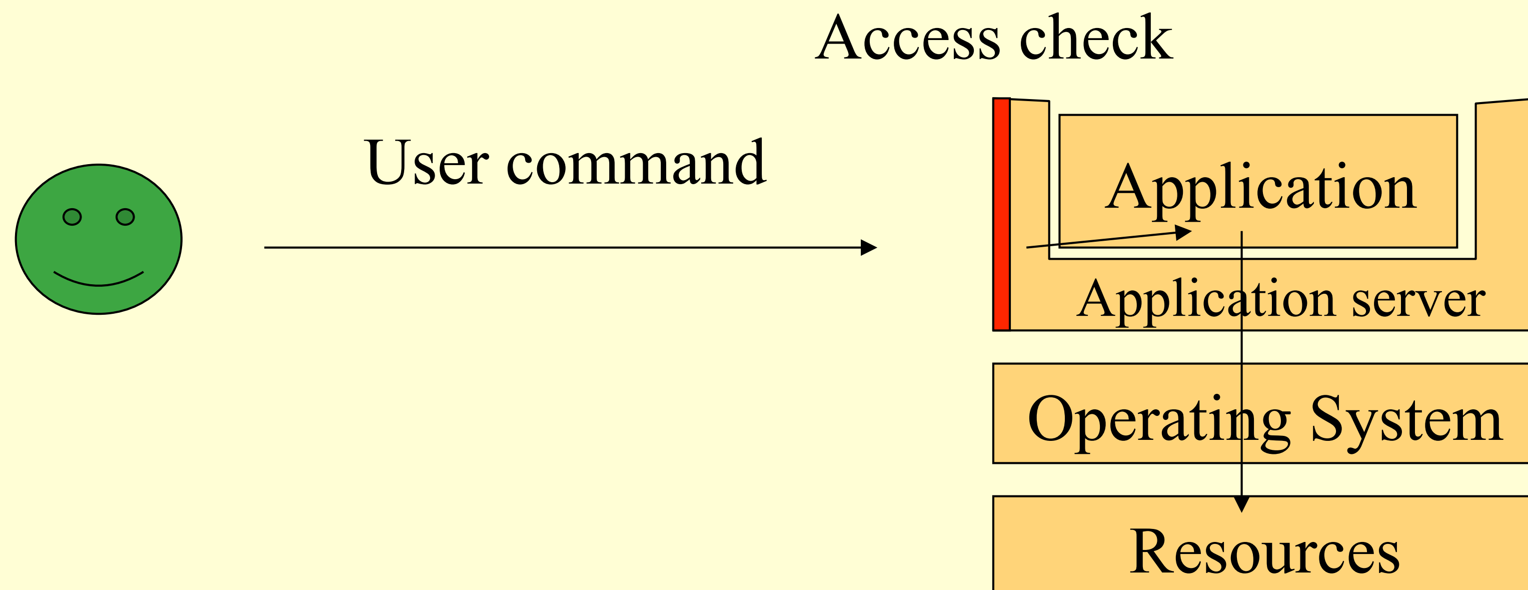
Approach #1: delegate to the OS

- All modern operating system have a built-in access control system, usually DAC based.
- If application resources can be mapped to OS resources, the OS access control can be reused



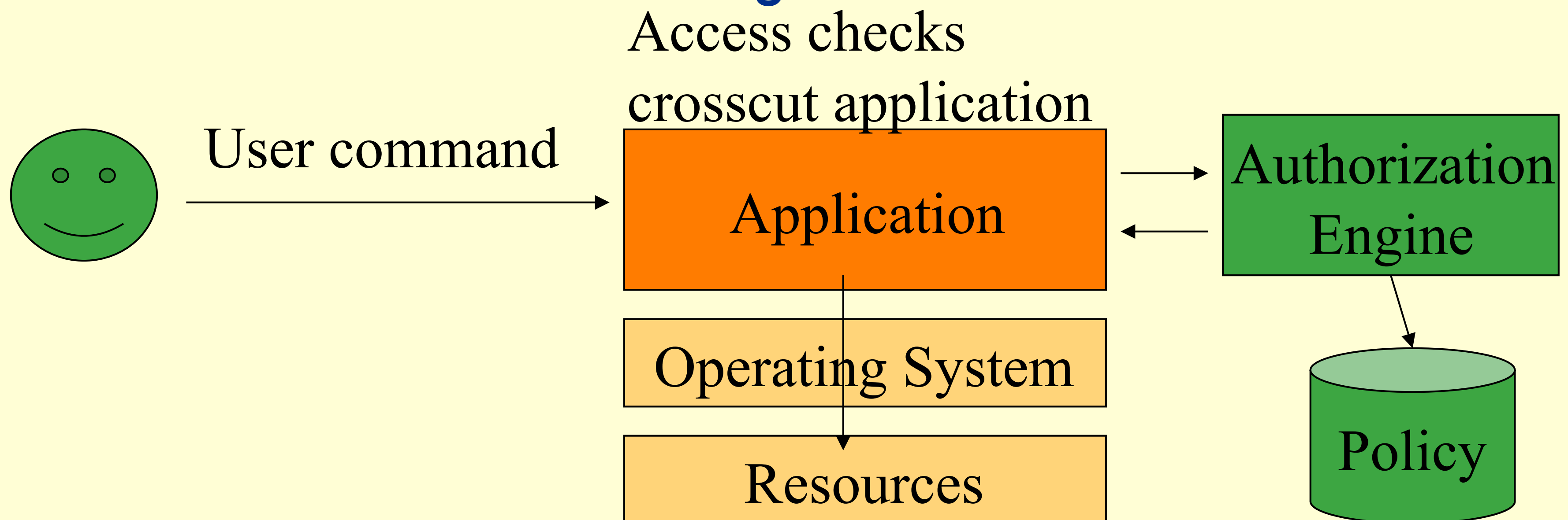
Approach #2: application servers

- Application server intercepts commands and performs access check
- E.g. J2EE and COM+, typically simple RBAC



Approach #3: in the application

- Application performs explicit checks in the application code
- It makes sense to externalize at least the policy to an authorization engine



OS Access Control

- In the rest of this session:
  - Helicopter overview of the Windows security architecture
  - Access control system in Windows
  - A brief look at Windows' implementation of the Biba Model.

Windows Access Control

- *Principals* are users or machines
  - Identified by Security Identifiers (SID)'s
    - E.g. S-1-5-21-XXX-XXX-XXX-1001
    - Hierarchical and globally unique
- *Authorities* manage principals and their credentials
  - Local Security Authority on each PC
  - Domain controller is authority for a domain
- Flexible mechanisms for slowly growing

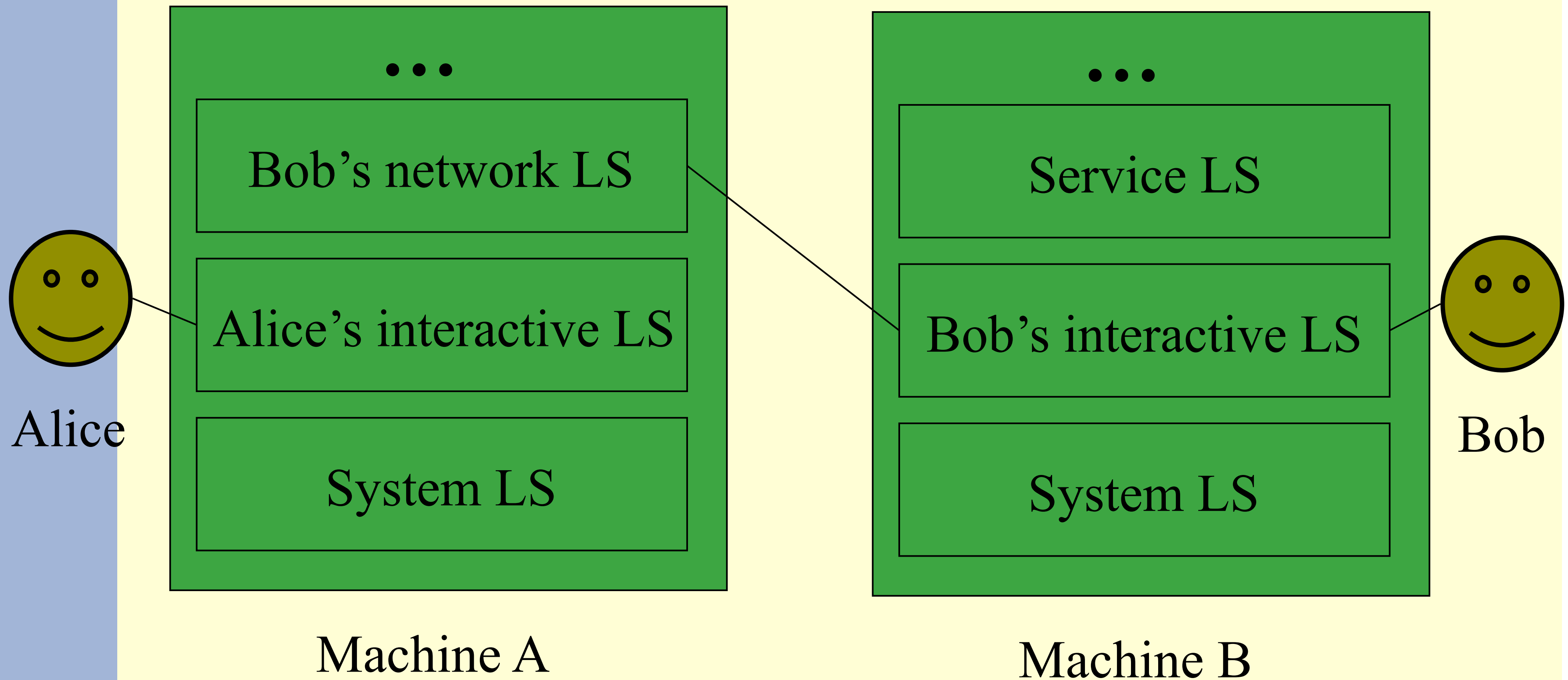
Windows Access Control

- *Trust* between authorities
  - Machine that is part of a domain trusts the domain
  - Domains can establish trust links
- *Authentication*
  - Via local password check on a standalone machine
    - Customizable via GINA
  - Via Kerberos or NTLM on a machine that is part of a domain
    - Customizable via SSPI

Windows Access Control

- Successful authentication leads to the creation of a *logon session*
  - Different types of logon sessions, e.g.
    - Interactive logon session, for a user that logs on locally
    - Network logon session, for a user that logs on remotely
    - Service logon session, for a service running as a given user
  - Logon session gets an *access token* that contains all *authorization attributes* for the user
- Processes and threads created in the logon session by default inherit the access token

Machines and logon sessions



Windows Access Control

- *Securable objects* include:
  - files, devices, registry keys, shared memory sections, …
- Every securable object carries a *security descriptor*, including a.o. an ACL.

Windows Access tokens

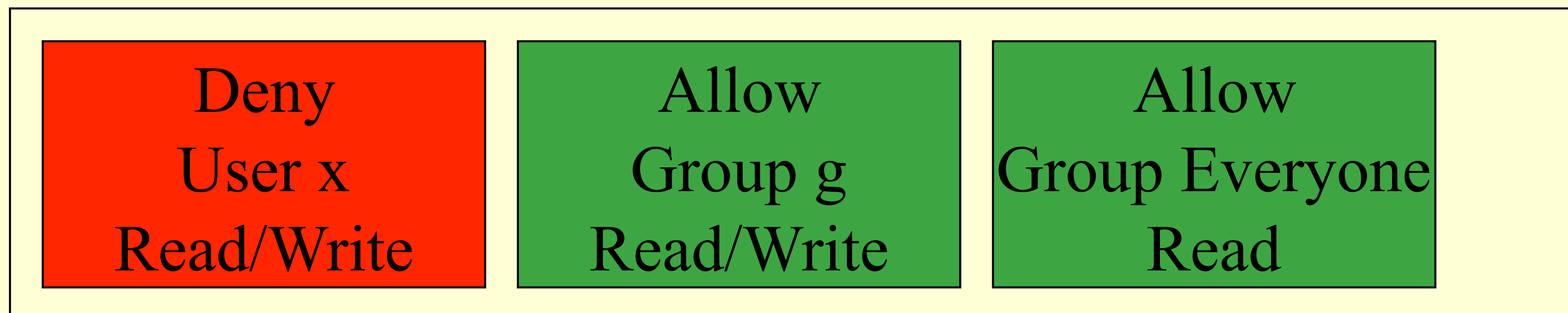
- Contain:
  - SID for the user
  - SID’s for the groups a user belongs to
    - Defined by the authority (typically domain)
    - Should reflect organizational structure
  - SID’s for the local groups (aliases) a user belongs to
    - Defined locally
    - Should reflect logical roles of applications on this machine
  - Privileges of the user, e.g.
    - Shutdown machine
    - Take ownership privilege (e.g. for Administrators)

Windows security descriptors

- Contain:
  - Owner SID
  - (Primary group SID)
  - DACL (Discretionary ACL): the ACL used for access control
  - SACL (System ACL): ACL specifying what should be audited
- Created at object creation time from a default template attached to the creating process

Windows DACL's

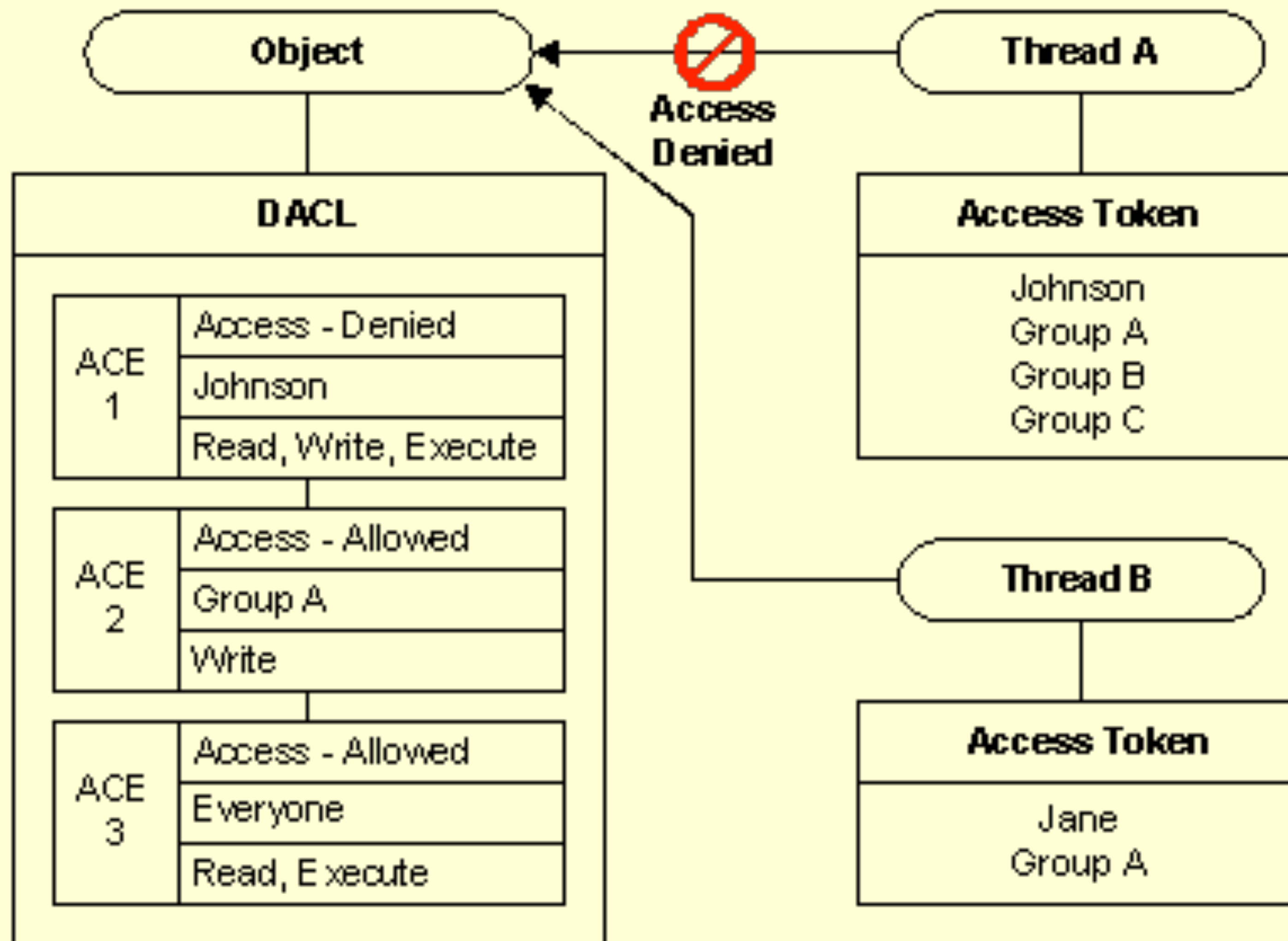
- A DACL contains a sorted list of access control entries
- Each access control entry denies or grants specific access rights to a group or user
- Access control entries that deny access should be placed in front of the list



Windows access control

- The kernel performs access checks for each securable object by:
  - Iterating over the access control entry in the DACL of the object
  - Each access control entry is matched to the access token of the accessing thread
  - The first match decides (hence deny entries should be before allow entries!)

Example



(Example from MSDN Library documentation)

Caching mechanisms

- Extensive caching is used to boost performance
  - Access token caches authorization attributes
  - Once a file is opened, the file handle is used as a capability, and no further access checks occur
    - Such a handle can be passed to other users
- Hence policy changes are not effective immediate if the affected user is currently logged on

Access control in applications

- Impersonation:
  - Server authenticates client, and puts access token on the thread servicing the request
- Role-based
  - Look for a local group SID corresponding to a role in the client access token
  - COM+ provides extensive support for this approach
- Object-based
  - Use an API for managing ACL's yourself

Running Least Privilege

- The OS Access Control system can also be used to “sandbox” applications to protect against:
  - Exploits of server programs
  - Trojans / viruses / bugs in any application
- Writing software to run in low-privileged accounts requires attention to:
  - What secured objects the application accesses
  - What privileged API’s the application uses

Windows access control

- Summary:
  - Access control based on:
    - Discretionary ACL’s
    - Privileges (safer than Unix root level access)
  - Protected operations depend on the type of object
  - Access control only performed during “opening” of an object. If access is granted, the opening process gets a capability for the requested access rights
  - RBAC can be simulated using local groups, but:
    - No sessions with limited activation of roles
    - Permissions associated with a role are spread over ACL’s

Windows Integrity Protection

- Windows Vista and later add a lattice-based access control model
  - But used for **integrity** control (as the Biba model)
- Securable objects get an *integrity level*
  - representing how important their integrity is
- Access Tokens get an *integrity level*
  - Representing how “contaminated” they are
- Three levels are distinguished:
  - High (admin), medium (user), low (untrusted)

Overview

- Introduction: Lampson's model for access control
- Classical Access Control Models
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-Based Access Control (RBAC)
  - Other Access Control Models
- Access Control in Windows
- Conclusion

Conclusion

- Most access control mechanisms implement the Lampson model
  - Principal – Action –Guard – Protected system
- Three important categories of access control policy models each have their own area of applicability
  - DAC in operating systems
  - RBAC in applications and databases
  - LBAC starting to find its use for integrity protection