Access control

Frank Piessens

(Frank.Piessens@cs.kuleuven.be)

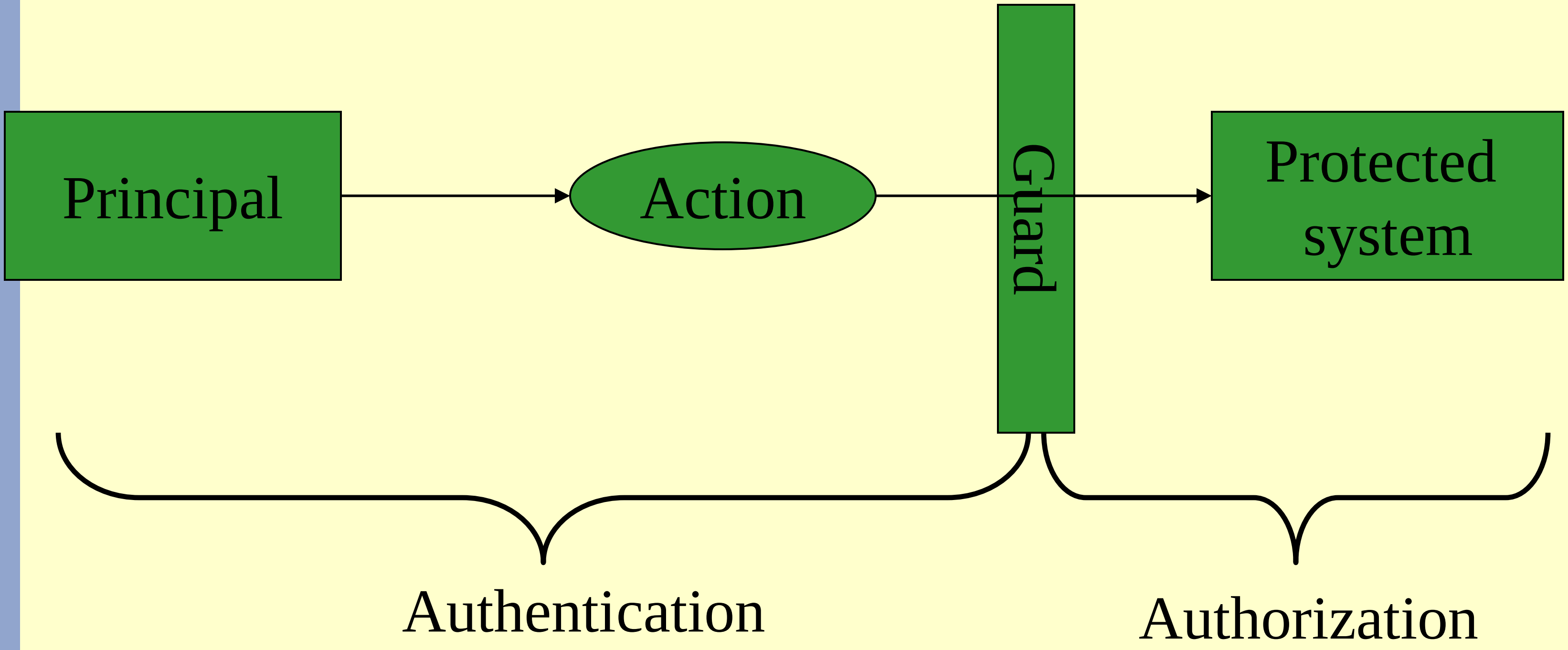
Overview

- Introduction: Lampson’s model for access control
- Classical User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- 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 program (e.g. An app on a Smartphone)
  - 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

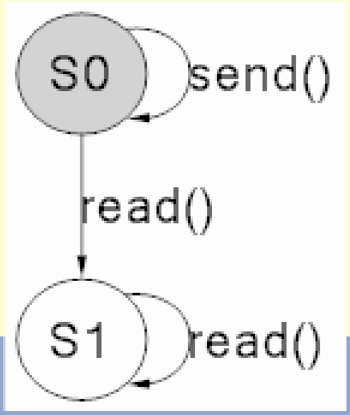
Authorization by the Guard

- 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
  - This includes: who is allowed to change the rules?
  - 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 User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- 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[<u’,o,r>] = true;
}
void deleteRight(User u, Right <u’,o,r>)
  requires (u in users) && (u’ in users) && (o in objects) && ownerOf[o] == u; {
    rights[<u’,o,r>] = false;
}
```

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

Overview

- Introduction: Lampson’s model for access control
- Classical User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- 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]}; {
}
```

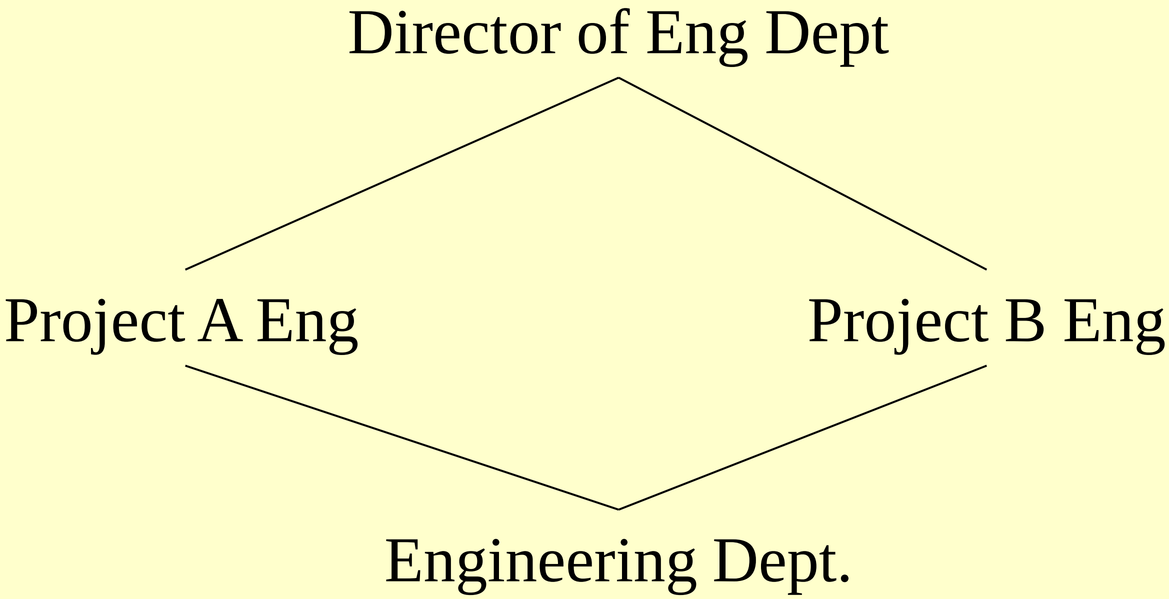
Security automaton for RBAC (ctd)

```
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 User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- Conclusion

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
- Authentication makes sure that every process / thread runs with an *access token* containing authorization attributes

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

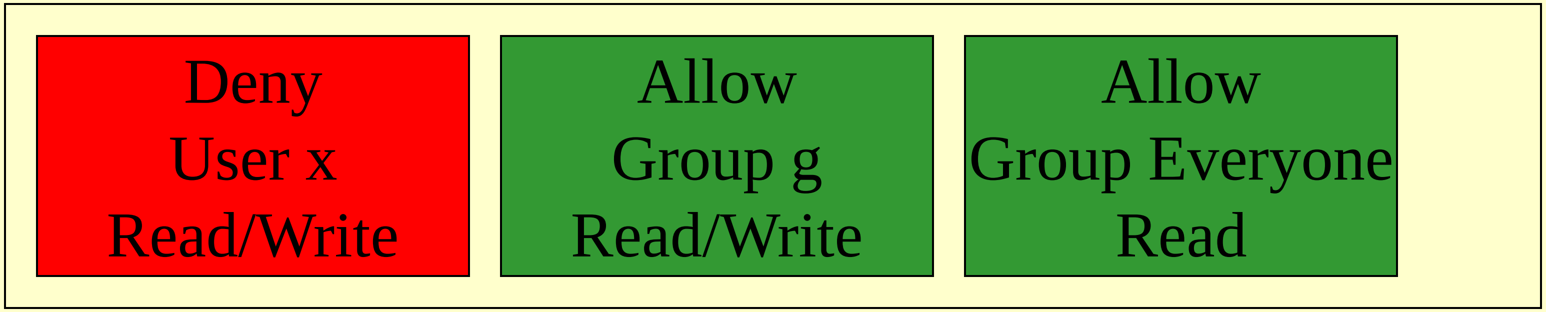
- Contain:
  - SID for the user
  - SID's for the groups a user belongs to
    - Defined by the authority (typically domain)
    - Should reflect organizational structure
  - SID's for the local groups (aliases) a user belongs to
    - Defined locally
    - Should reflect logical roles of applications 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



Deny
User x
Read/Write

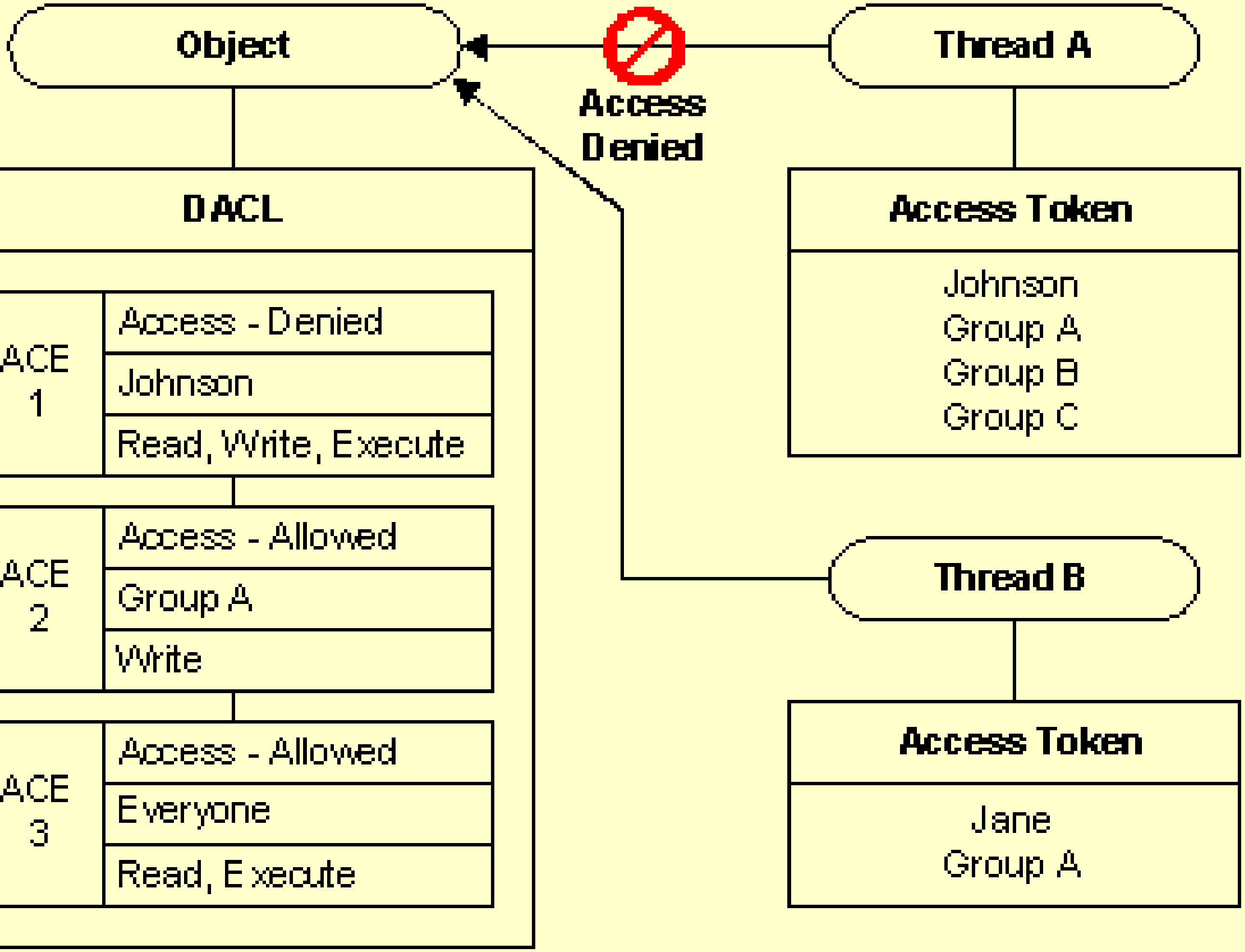
Allow
Group g
Read/Write

Allow
Group Everyone
Read

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



Object

Access Denied

Thread A

DACL

ACE 1

Access - Denied

Johnson

Read, Write, Execute

ACE 2

Access - Allowed

Group A

Write

ACE 3

Access - Allowed

Everyone

Read, Execute

Access Token

Johnson
Group A
Group B
Group C

Thread B

Access Token

Jane
Group A

(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

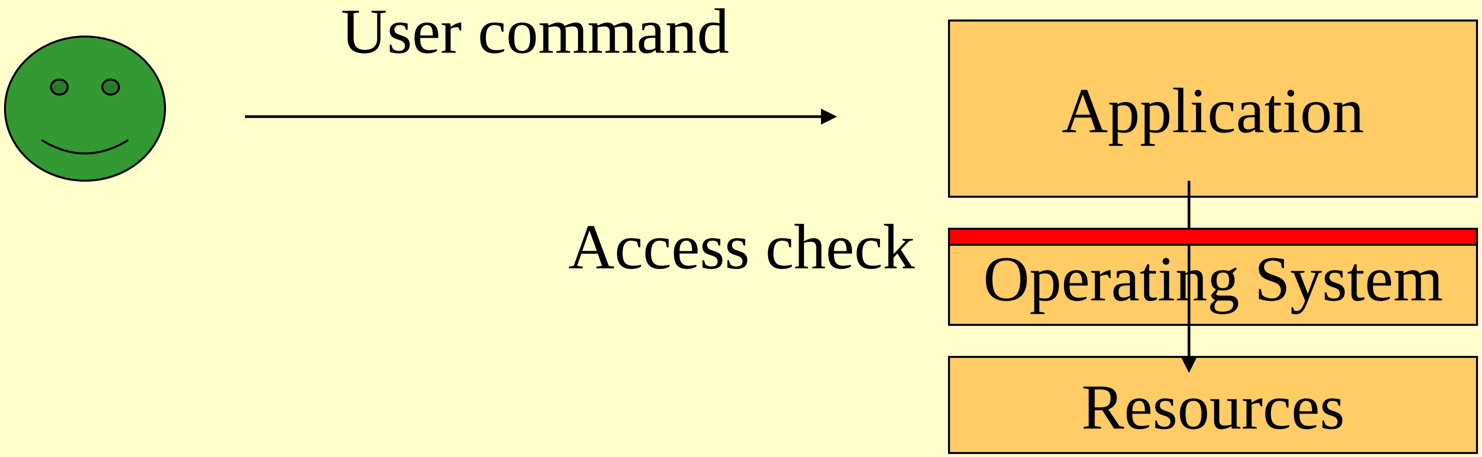
Implementing Access Control in Applications

- Several options

1. Delegate to OS
2. Rely on application server
3. Enterprise security middleware
4. Roll your own

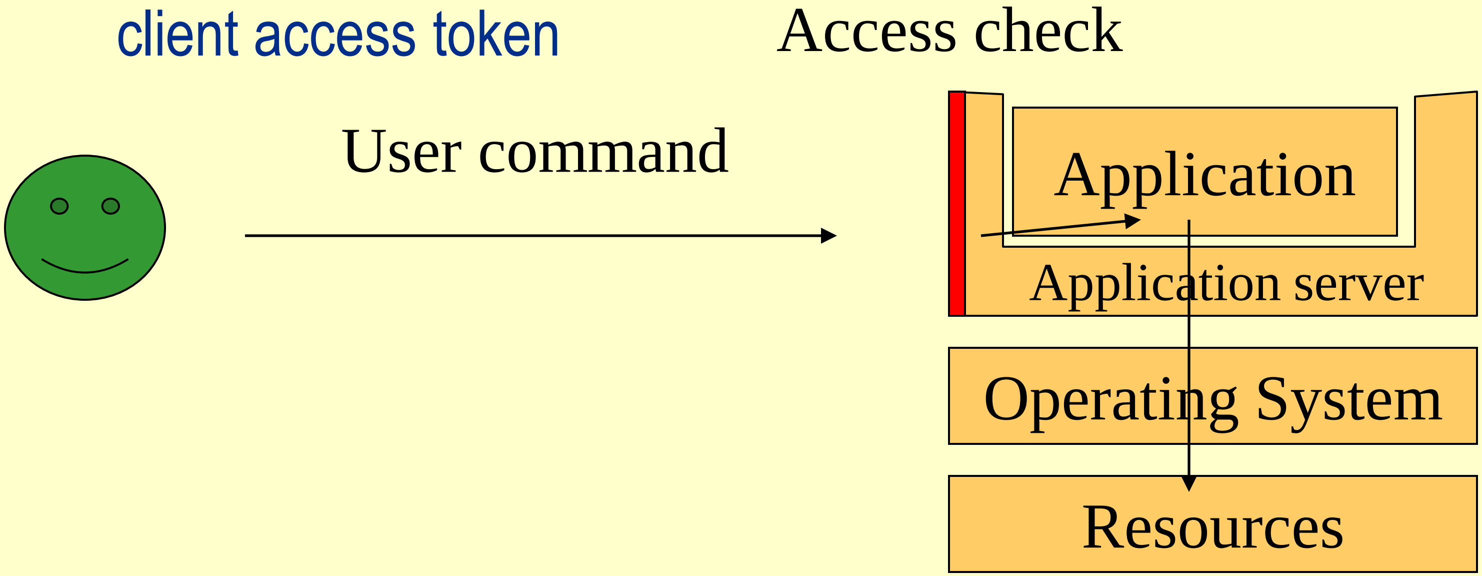
Approach #1: delegate to the OS

- If application resources can be mapped to OS resources, the OS access control can be reused
- E.g. in Windows:
  - Server authenticates client, and puts access token on the thread servicing the request



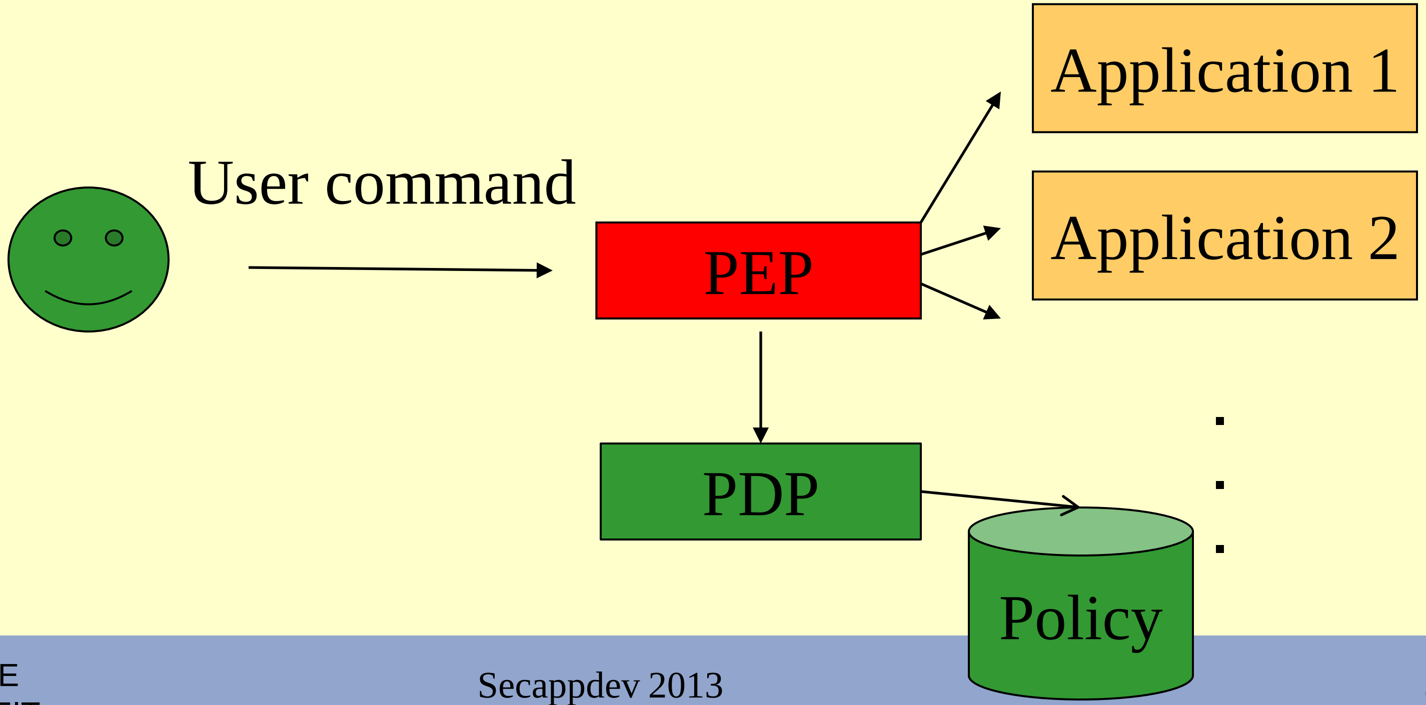
Approach #2: application servers

- Application server intercepts commands and performs access check
- E.g. in Windows COM+:
  - Look for a local group SID corresponding to a role in the client access token



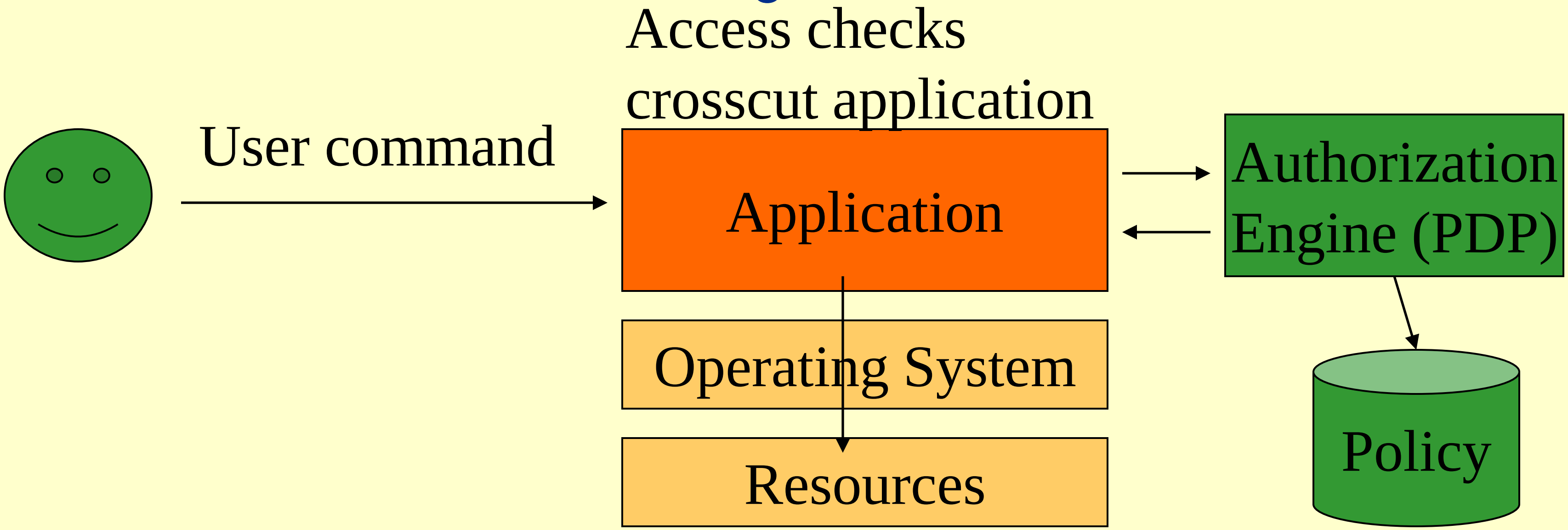
Approach #3: security middleware

- Reverse proxy intercepts commands and performs access check
- E.g. IBM WebSEAL



Approach #4: in the application

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



Overview

- Introduction: Lampson’s model for access control
- Classical User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- Conclusion

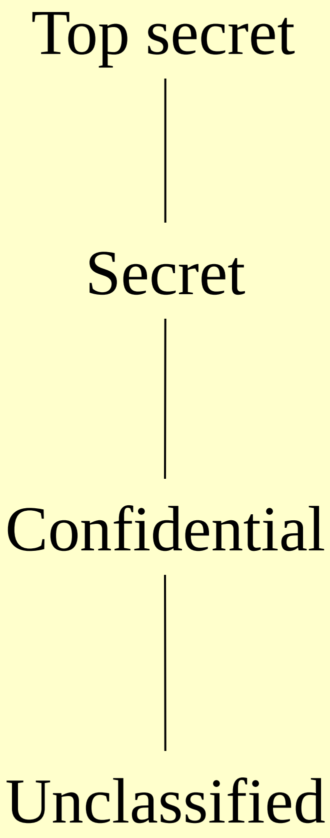
Introduction

- If the software that a user is running can not be trusted, access control is more complicated
  - E.g. Trojan horses
  - E.g. Smartphone apps, Web gadgets, …
- Additional issues include:
  - How can you give SW access to information, but limit what the SW can do with that information
    - Usage control / information flow control
  - The confused deputy problem

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

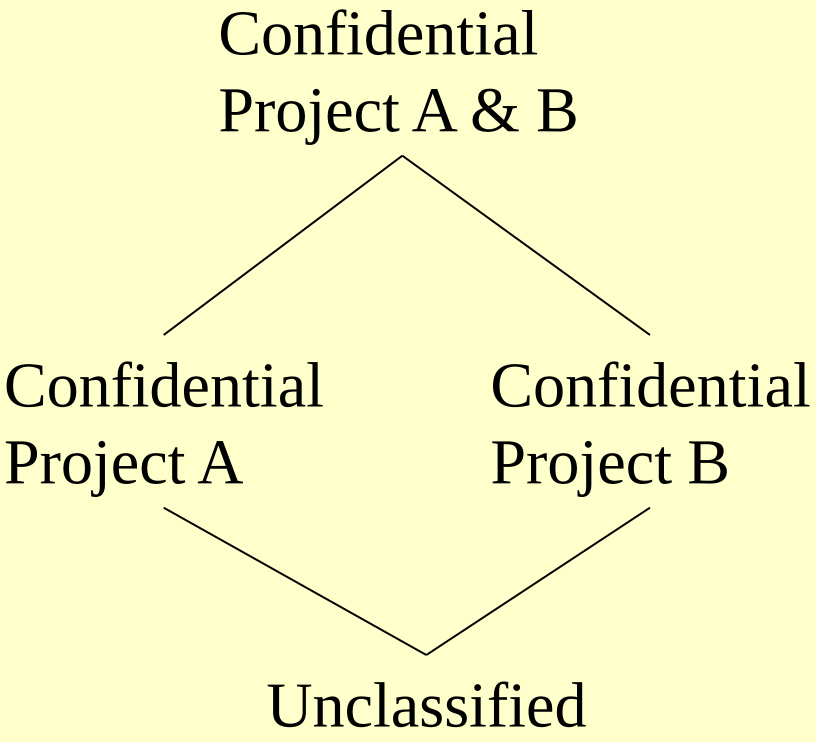


Top secret

Secret

Confidential

Unclassified



Confidential
Project A & B

Confidential
Project A

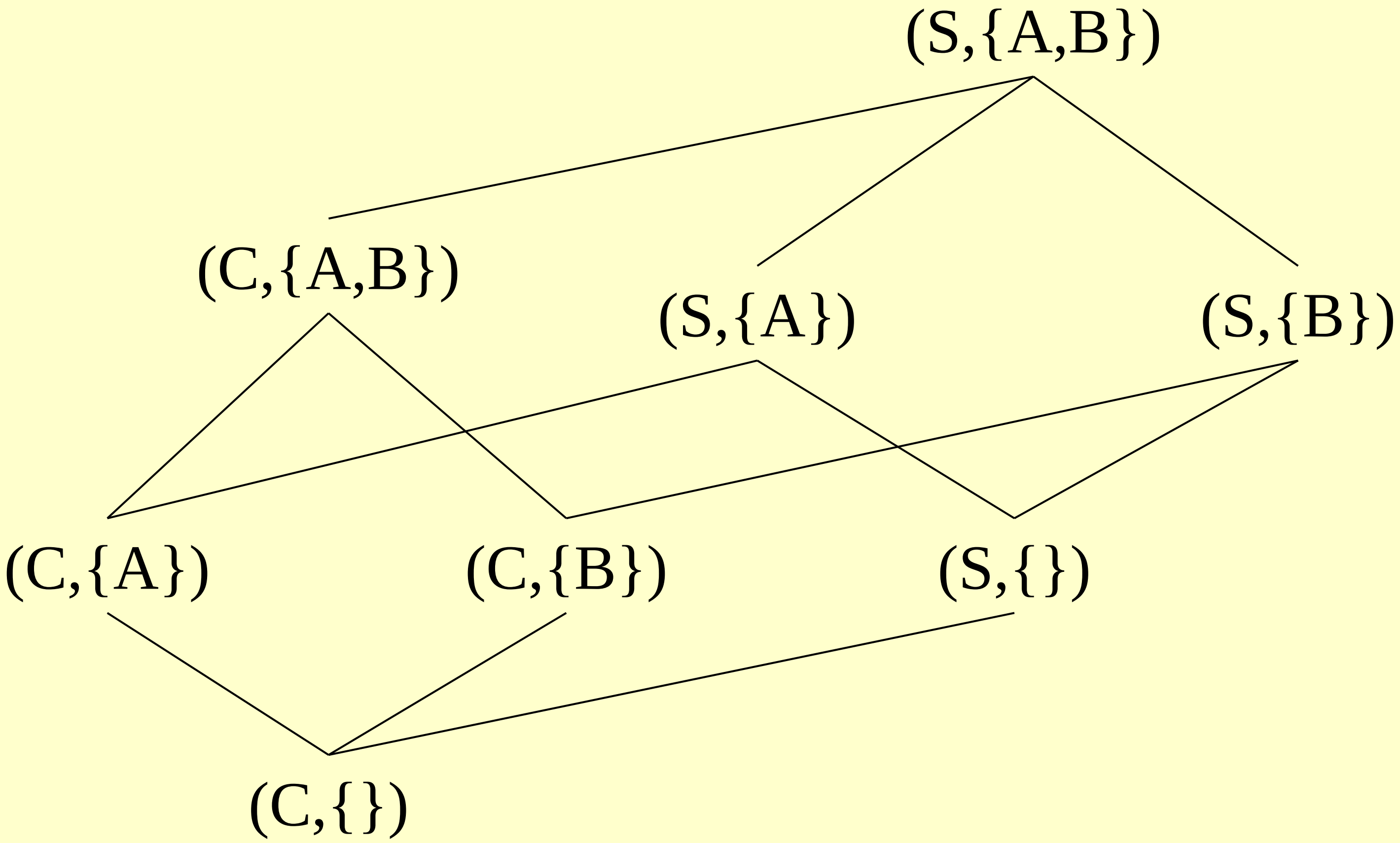
Confidential
Project B

Unclassified

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



(S,{A,B})

(C,{A,B})

(S,{A})

(S,{B})

(C,{A})

(C,{B})

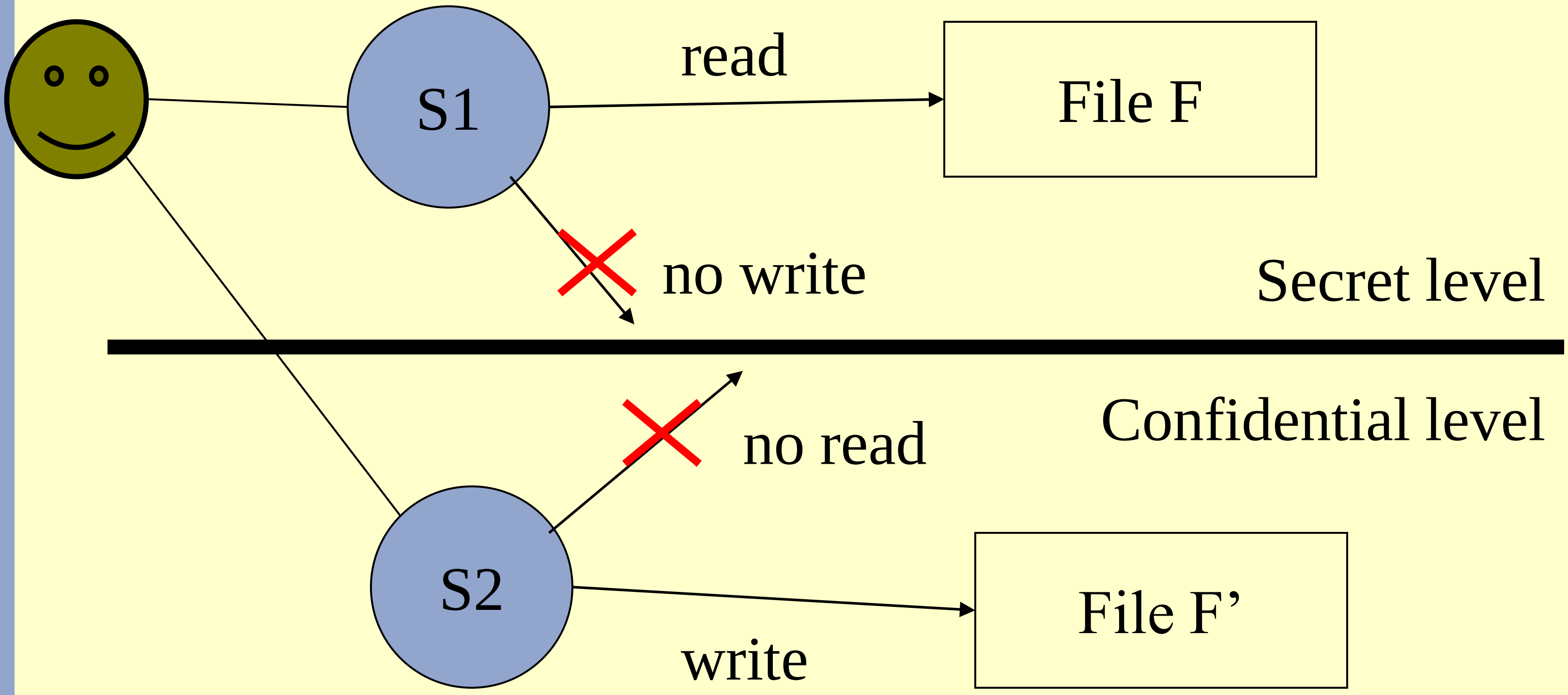
(S,{})

(C,{})

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
- But LBAC is used in practice for addressing integrity concerns rather than confidentiality concerns

Windows Integrity Protection

- Windows Vista and later add a lattice-based access control model
  - But used for **integrity** control (this dual interpretation of LBAC is called 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

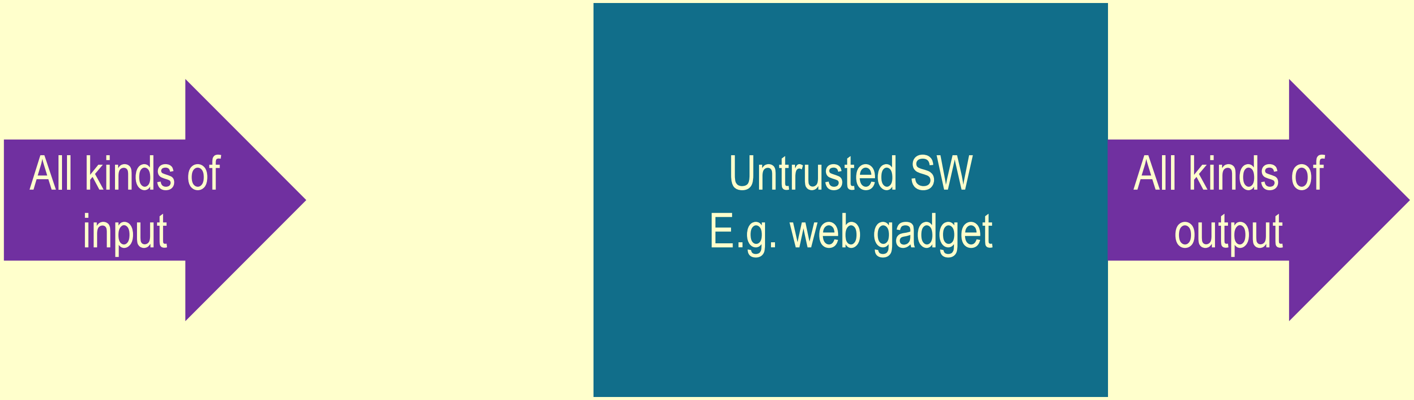
- Introduction: Lampson’s model for access control
- Classical User Access Control Models
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Implementation techniques
- Access Control for Untrusted Software
  - Mandatory Access Control (MAC)
  - Usage Control and Information Flow Control
  - Implementation techniques
- Conclusion

Introduction

- Given the problems with LBAC but the importance of containing untrusted software, researchers are studying alternative techniques:
  - Usage control: how can one give access to resources but limit how they are used
  - Information flow control: how can one give access to information but limit how it can be disseminated
    - LBAC is a very rough approximate solution for this

Example: Information flow control

- Information flow control is a class of technical countermeasures that try to enforce that software can not leak information – not even indirectly!



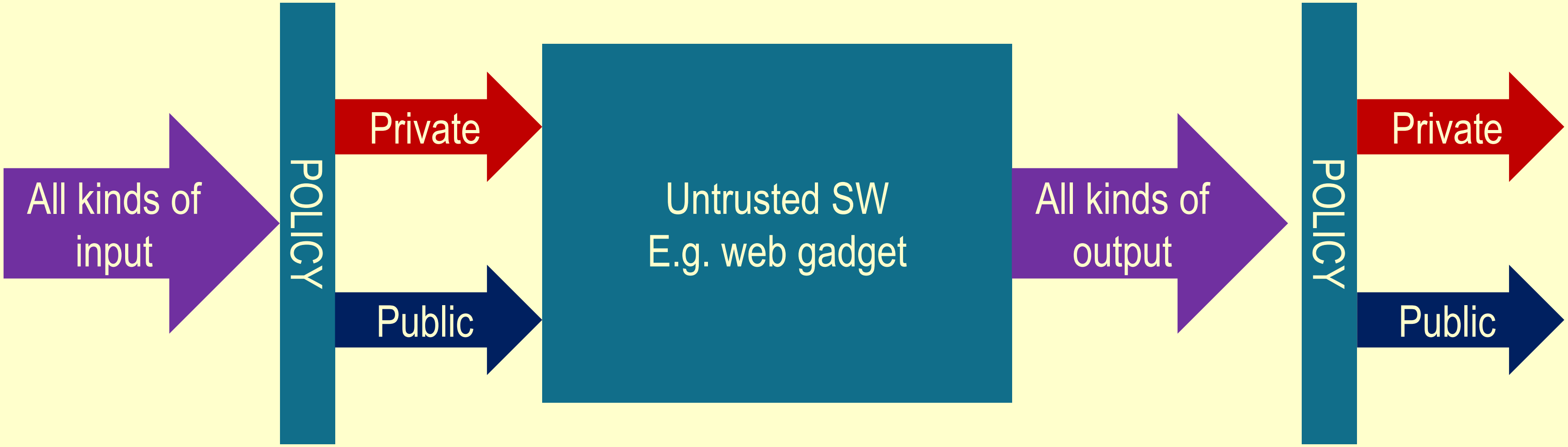
All kinds of input

Untrusted SW
E.g. web gadget

All kinds of output

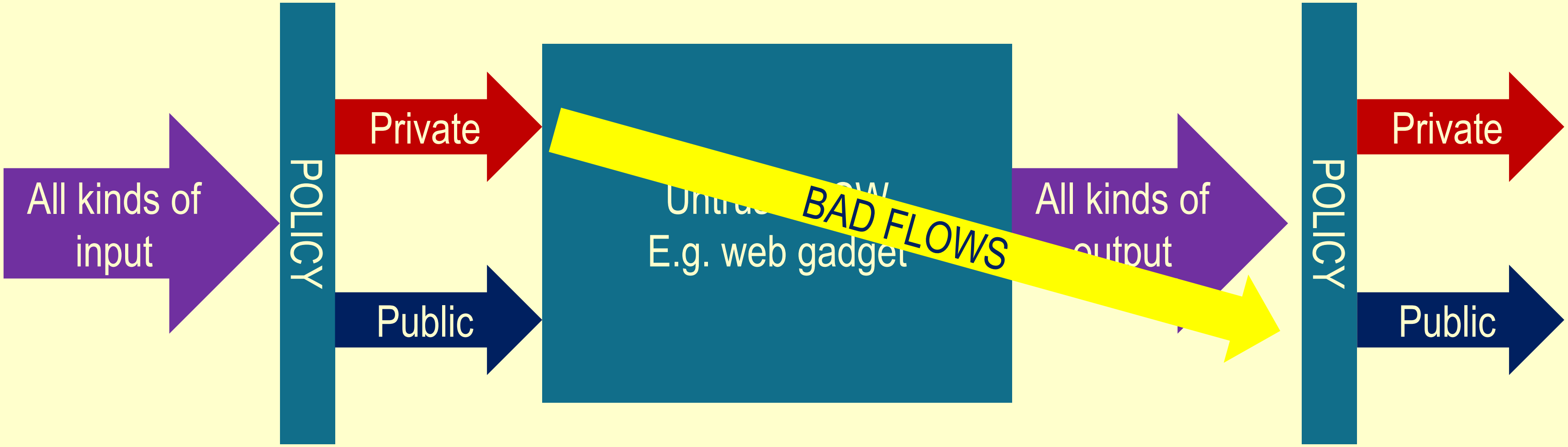
Example: Information flow control

- Information flow control is a class of technical countermeasures that try to enforce that software can not leak information – not even indirectly!



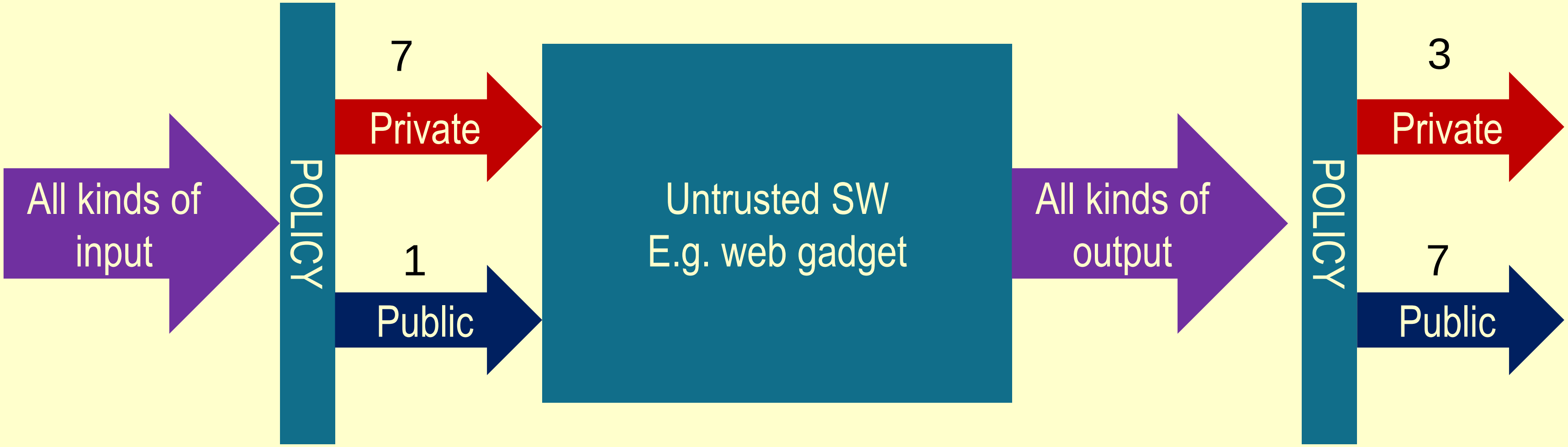
Example: Information flow control

- Information flow control is a class of technical countermeasures that try to enforce that software can not leak information – not even indirectly!



Information flow control

- IFC can not be enforced precisely by runtime monitoring alone



Secure:
```
Out_low := In_low + 6
```

Insecure:
```
Out_low := In_high
```

Insecure:
```
if (In_high > 10) {
    Out_low := 3;
}
else Out_low := 7
```

Example: information flow control in Javascript

- Modern web applications use client-side scripts for many purposes:
  - Form validation
  - Improving interactivity / user experience
  - Advertisement loading
  - ...
- Malicious scripts can enter a web-page in various ways:
  - Cross-site-scripting (XSS)
  - Malicious ads
  - Man-in-the-middle
  - ...

Example: information flow control in Javascript

HIGH INPUT

```
var text = document.getElementById('email-input').text;
var abc = 0;

if  (text.indexOf('abc') != -1)
   { abc = 1 };

var url = 'http://example.com/img.jpg' + '?t=' + escape(text) + abc;

document.getElementById('banner-img').src = url;
```

LOW OUTPUT

Example: information flow control in Javascript

HIGH INPUT

```
var text = document.getElementById('email-input').text;
var abc = 0;

if  (text.indexOf('abc') != -1)
   { abc = 1 };

var url = 'http://example.com/img.jpg' + '?t=' + escape(text) + abc;

document.getElementById('banner-img').src = url;
```

Explicit flow

Implicit flow

LOW OUTPUT

Enforcement mechanisms

- Static, compile-time techniques
  - Classify (=type) variables as either high or low
  - Forbid:
    - Assignments from high expressions to low variables
    - Assignments to low variables in “high contexts”
    - ...
- Two mature languages (research prototypes):
  - Jif: a Java variant
  - FlowCaml: an ML variant
- Experience: quite restrictive, labour intensive
  - Probably only useful in high-security settings

Enforcement mechanisms

- Runtime techniques
  - Approximate non-interference with a safety property
  - Label all data entering the program with an appropriate security level
  - Propagate these levels throughout the computation
  - Block output of high-labeled data to a low output channel
- Several mature and practical systems, but all with (some) remaining holes
- Some sound systems, but quite expensive

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
- Researchers are looking into ways to enforce more fine-grained policies in the presence of untrusted software