



4. Access Control

Computer Security Courses @ POLIMI

What is Access Control?

- A binary decision:
 - Access either *allowed* or *denied*
 - What could *possibly* go wrong?
- Scale goes wrong!
 - You cannot explicitly *list* the answers
 - Need to condense them in *rules*
- Questions
 - How do we *design* the access rules?
 - How do we *express* the access rules in practice?
 - How do we appropriately *apply* them?

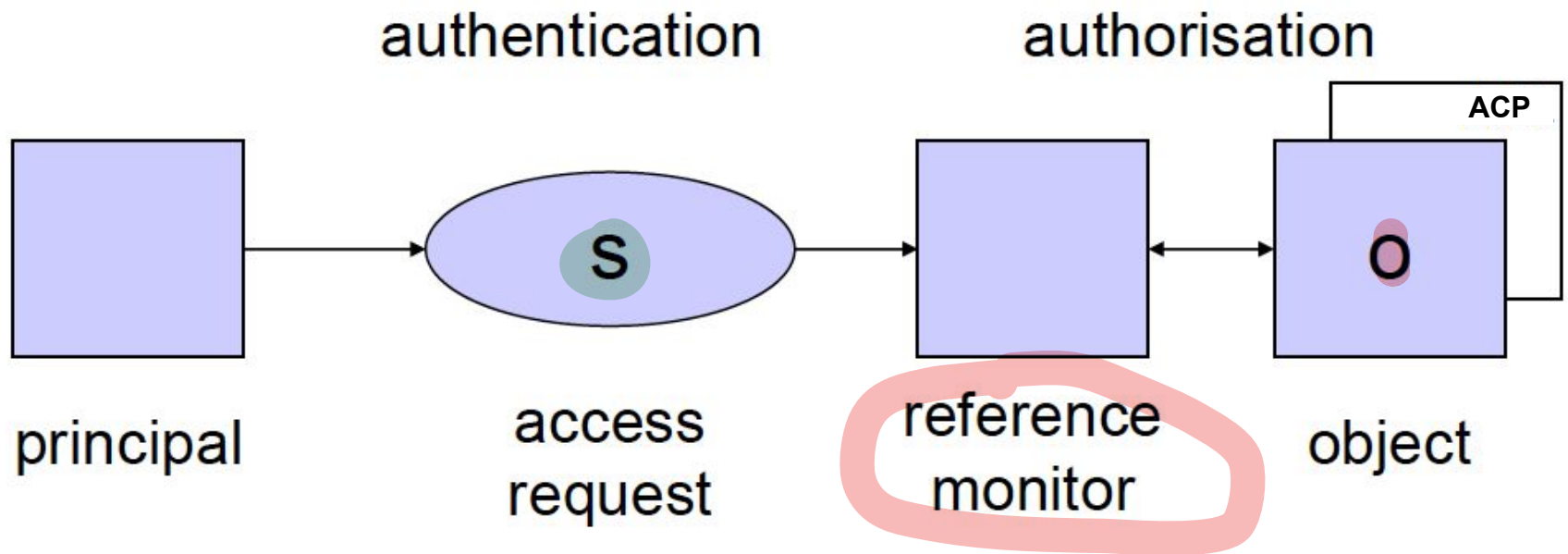
Who Does it? The Reference Monitor

Enforces access control policies ("who does what on which resource"). All modern kernels have a reference monitor implementation.

Requirements for the RM:

- Tamper proof
- Cannot be bypassed
- Small enough to be verified/tested

Authentication & Authorization




Authentication & Authorization

- **Authentication:** reference monitor verifies the identity of the principal (user) making the request:
 - User enters username and password.
 - If the values entered are correct, the user is “authenticated”.
 - We could say: “The machine now runs on behalf of the user”.
 - This might be intuitive, but it is imprecise.
 - Log on creates a **process** that runs with access rights assigned to the user.
 - Typically, the process runs under the user identity of the user who has logged on.

Authentication & Authorization

- **Authentication:** reference monitor verifies the identity of the principal (user) making the request.
 - **Authorization:** reference monitor decides whether access is granted or denied.
 - Reference monitor has to find and evaluate the security policy relevant for the given request.
- “Easy” in centralized systems but in distributed systems,
- how to find all relevant policies?
 - how to make decisions if policies may be missing?

Users & User Identities

- Requests to reference monitor do not come directly from a user or a user identity, but from a process.
- The active entity making a request within the system is called the subject.
- You must distinguish between these concepts: 
 - **User:** person;
 - **Principal (User identity):** name used in the system, possibly associated with a user; Active entity.
 - **Subject: Process** running under a given user identity.
 - **Object:** Passive entity – file or resource.
 - **Access operations:** Vary from basic memory access (read, write) to method calls in object-oriented systems.

Access control models

Can be roughly divided in



- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Role-Based Access Control (RBAC)

Difference between DAC and MAC

- who assigns privileges

RBAC abstracts roles from identities

Discretionary Access Control

- Resource owner *discretionarily* decides its access privileges
 - Stefano creates a file
 - Assigns Federico the privilege of reading it
- If this sounds “normal” it is because **all** off-the-shelf OSs implement DAC
 - Windows
 - Linux and other UNIX flavors
 - Mac OS X
 - Also applications, social networks...mostly DAC!

Examples of DAC systems

- UNIX

- **Subjects:** users, groups
- **Objects:** files (everything, really)
- **Actions:** read, write, execute

- Windows (not the 95/98/ME branches)

- **Subjects:** with *roles* instead of *groups*, multiple ownership of users and roles over files
- **Objects:** files and “other” resources
- **Actions:** delete, change permissions, change ownership.

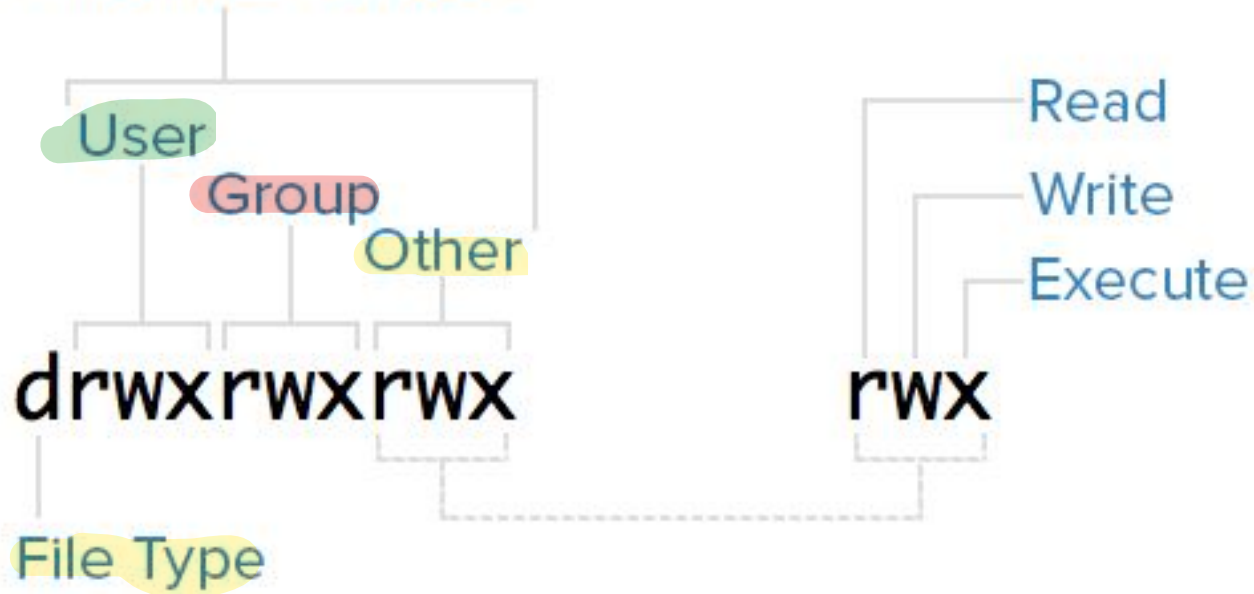
Permissions (r, w, x)	Subjects (user, group)							Objects (files, dirs, ...)
lrwxr-xr-x	1	phretor	staff	30	Jul	3	2011	.irssi -> /Users/ph
-rw-r--r--	1	phretor	staff	140	Dec	1	2012	.jackdrc
drwxr-xr-x	3	phretor	staff	102	Feb	21	2008	.jmf
-rwxrwxrwx@	1	phretor	staff	41	Dec	9	2012	.khlbshcrc
-rw-----	1	root	staff	51	Oct	27	2008	.lessht
drwxr-xr-x	4	phretor	staff	136	Jan	4	2008	.lftp
drwx-----	4	phretor	staff	136	Jul	22	2009	.links
drwxr-xr-x	3	phretor	staff	102	Apr	1	2013	.m2
lrwxr-xr-x	1	phretor	staff	32	Jul	3	2011	.mailcap -> /Users/
drwxr-xr-x	3	phretor	staff	102	Jan	7	2008	.mldonkey
drwxr-xr-x	4	phretor	staff	136	Jan	25	16:29	.mono
lrwxr-xr-x	1	phretor	staff	31	Jul	3	2011	.mutt.d -> /Users/p
lrwxr-xr-x	1	phretor	staff	36	Jul	3	2011	.muttprintrc -> /Us
lrwxr-xr-x	1	phretor	staff	31	Jul	3	2011	.muttrc -> /Users/p
drwxr-xr-x	11	phretor	staff	374	Jan	31	2008	.ncftp
drwxr-xr-x	8	phretor	staff	272	Dec	7	19:59	.neocomplcache
drwxr-xr-x	8	phretor	staff	272	Oct	21	2012	.neocon
drwxr-xr-x	11	phretor	staff	374	Feb	9	2013	.npm
lrwxr-xr-x	1	phretor	staff	38	Jul	3	2011	.offlineimaprc -> /
drwxr-xr-x	15	phretor	staff	510	Feb	4	22:23	.oh-my-zsh
drwxr-xr-x	6	phretor	staff	204	Apr	20	2013	.parentseye
drwxrwxr-x	3	phretor	staff	102	Dec	24	2010	.pip
drwx-----	6	phretor	staff	204	Apr	5	2008	.psi
-rw-----	1	phretor	staff	90	Mar	17	2010	.psql_history

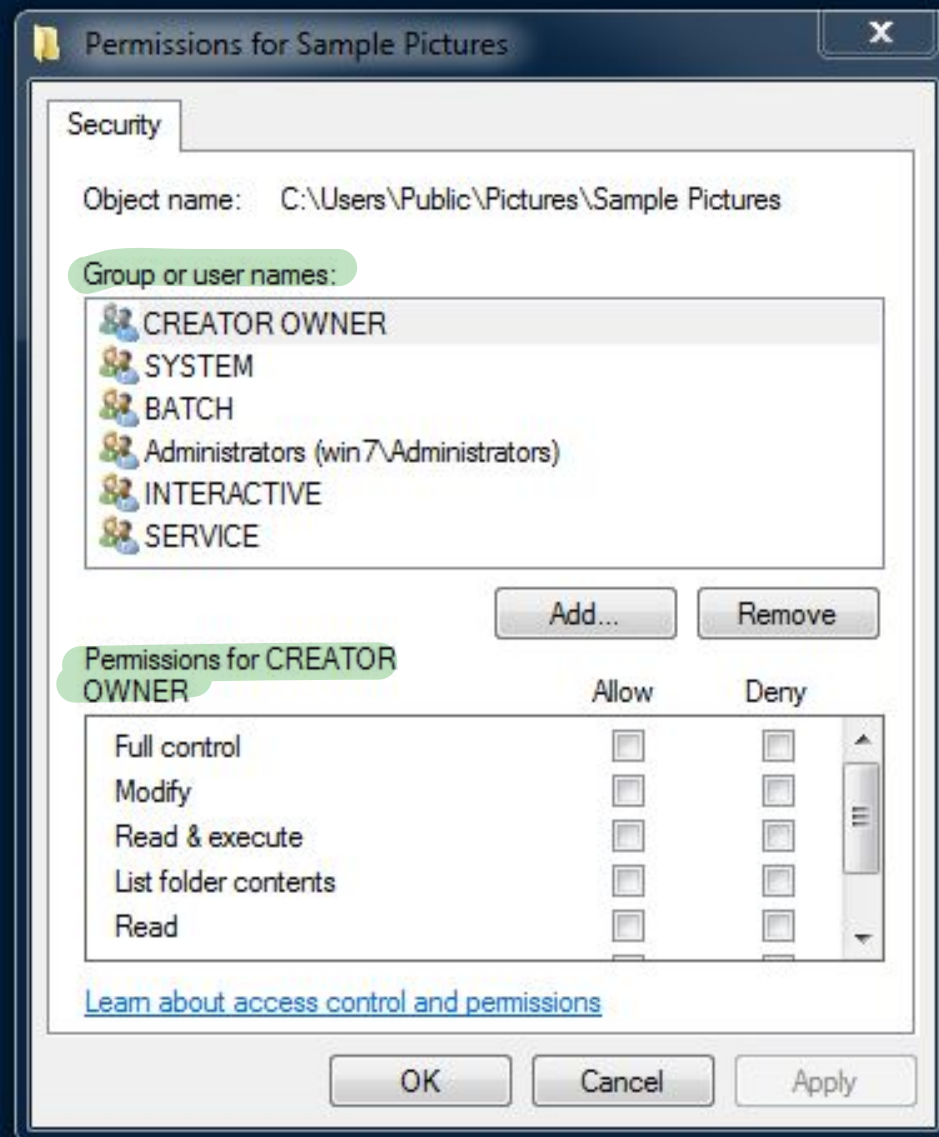
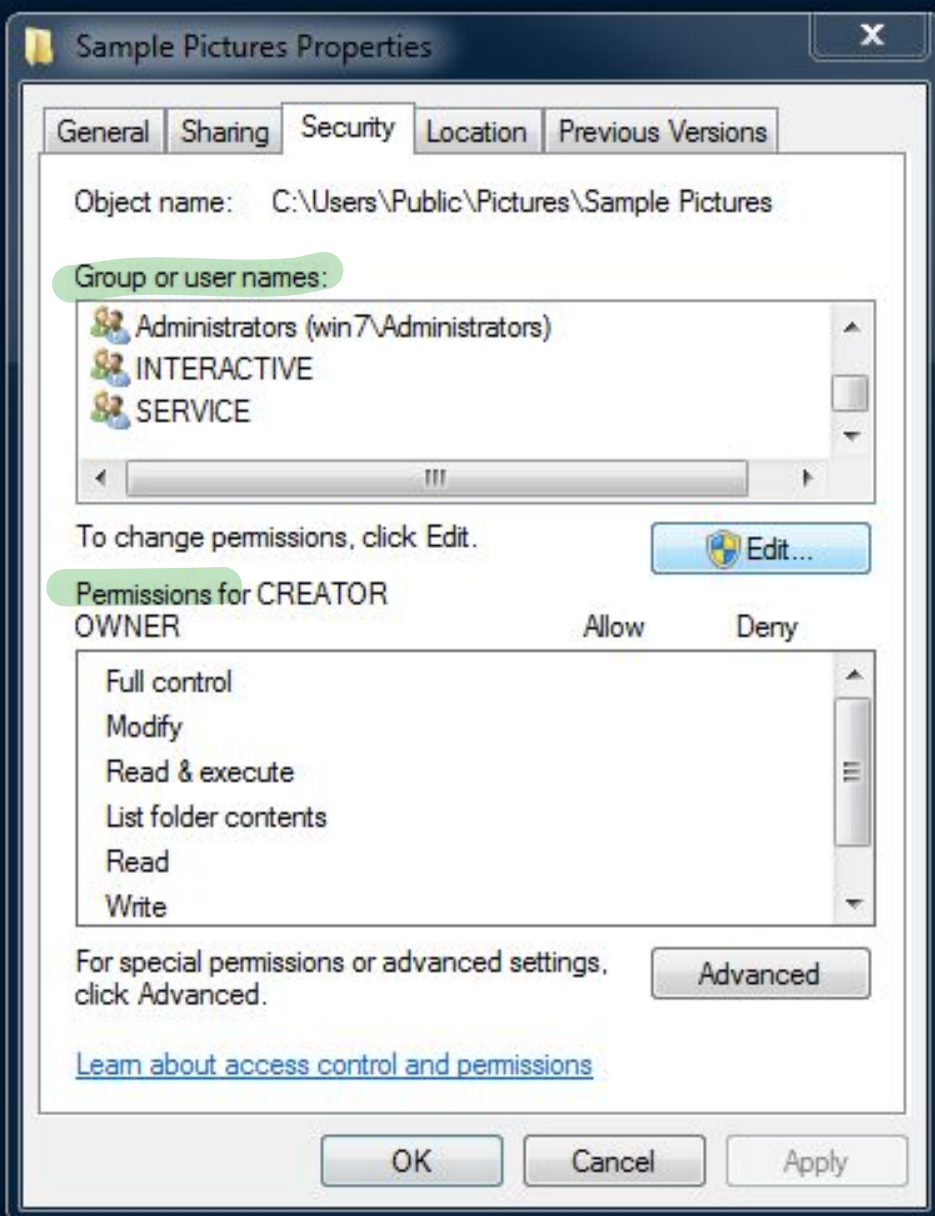
UNIX Permissions

Mode		Owner	Group	File Size	Last Modified			Filename
drwxrwxrwx	2	sammy	sammy	4096	Nov	10	12:15	everyone_directory
drwxrwx---	2	root	developers	4096	Nov	10	12:15	group_directory
-rw-rw----	1	sammy	sammy	15	Nov	10	17:07	group_modifiable
drwx-----	2	sammy	sammy	4096	Nov	10	12:15	private_directory
-rw-----	1	sammy	sammy	269	Nov	10	16:57	private_file
-rwxr-xr-x	1	sammy	sammy	46357	Nov	10	17:07	public_executable
-rw-rw-rw-	1	sammy	sammy	2697	Nov	10	17:06	public_file
drwxr-xr-x	2	sammy	sammy	4096	Nov	10	16:49	publicly_accessible_directory
-rw-r--r--	1	sammy	sammy	7718	Nov	10	16:58	publicly_readable_file
drwx-----	2	root	root	4096	Nov	10	17:05	root_private_directory

Permissions “Triads”



Permissions Classes





General model of DAC systems

- We need to model the following entities:
 - **Subjects** who can exercise privileges (a.k.a., rights)
 - **Objects** on which privileges are exercised
 - **Actions** which can be exercised
- **Protection state:** a triple (S, O, A)
 - A : matrix with S rows and O columns
 - $A[s,o]$: privileges of subject s over object o



	file1	file2	directoryX
Alice	Read	Read, Write, Own	
Bob	Read, Write, Own	Read	Read, Write, Own
Charlie	Read, Write		Read

Transitions in the HRU model

- **Basic operations**

- create (or destroy) subject $\langle s \rangle$
- create (or destroy) object $\langle o \rangle$
- add (or remove) $\langle \textit{permission} \rangle$ into $[s,o]$ matrix

- **Transitions:** sequences of basic operations

- *"create file (subject u ; file f)"*:
 - create object f
 - add "own" into $[u,f]$
 - add "read" into $[u,f]$
- Is this right?

Transitions in the HRU model

- **Basic operations**

- create (or destroy) subject $\langle s \rangle$
- create (or destroy) object $\langle o \rangle$
- add (or remove) $\langle permission \rangle$ into $[s, o]$ matrix

- **Transitions:** sequences of basic operations

- *"create file (subject u ; file f)"*:
 - create object f
 - add "own" into $[u, f]$
 - add "read" 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 for modeling transitions

Safety problems

- From an initial configuration, given a sequence of transitions, can s obtain a right r on f ?
 - Obviously, yes if the owner o allows it!
 - But, if the owner does not?
 - If it happens, set of commands **unsafe by design!**
- More formally
 - Given an initial protection state and set of transitions, is there **any sequence of transitions that leaks a certain right r** (for which the owner is removed) into the access matrix?
 - If not, then the system is safe with respect to right r
- In a generic HRU model (with infinite resources): **undecidable** problem
 - Decidable in mono-operational systems, (substantially useless, e.g., **you cannot create a file and own it**)
 - .. or if subjects/objects are finite.

Common DAC Implementations

- Reproduction of HRU models
- Access matrix is a sparse matrix
- Alternative implementations:
 - **Authorizations table:** records non-null triples S-O-A, typically used in DBMS
 - **Access Control Lists:** records by column (i.e., for each object, the list of subjects and authorizations)
 - **Capability Lists:** records by row (i.e., for each subject, the list of objects and authorizations)



Access Control vs Capability Lists

Access Control Lists

- Focus on the object
- ACLs \equiv columns of the access control matrix

fun.com	Alice: {exec}	Bill: {exec,read,write}
---------	---------------	-------------------------

Capability Lists



- Focus on the subject
- Capabilities \equiv rows of the access control

Alice	edit.exe: {exec}	fun.com: {exec,read}
-------	------------------	----------------------

ACLs vs Capability Lists

ACLs

- efficient with **per object** operations
- Most common case
- Some systems (e.g., POSIX) use abbreviated ACLs
- Cannot have multiple owners (partially achievable via groups).

Capabilities

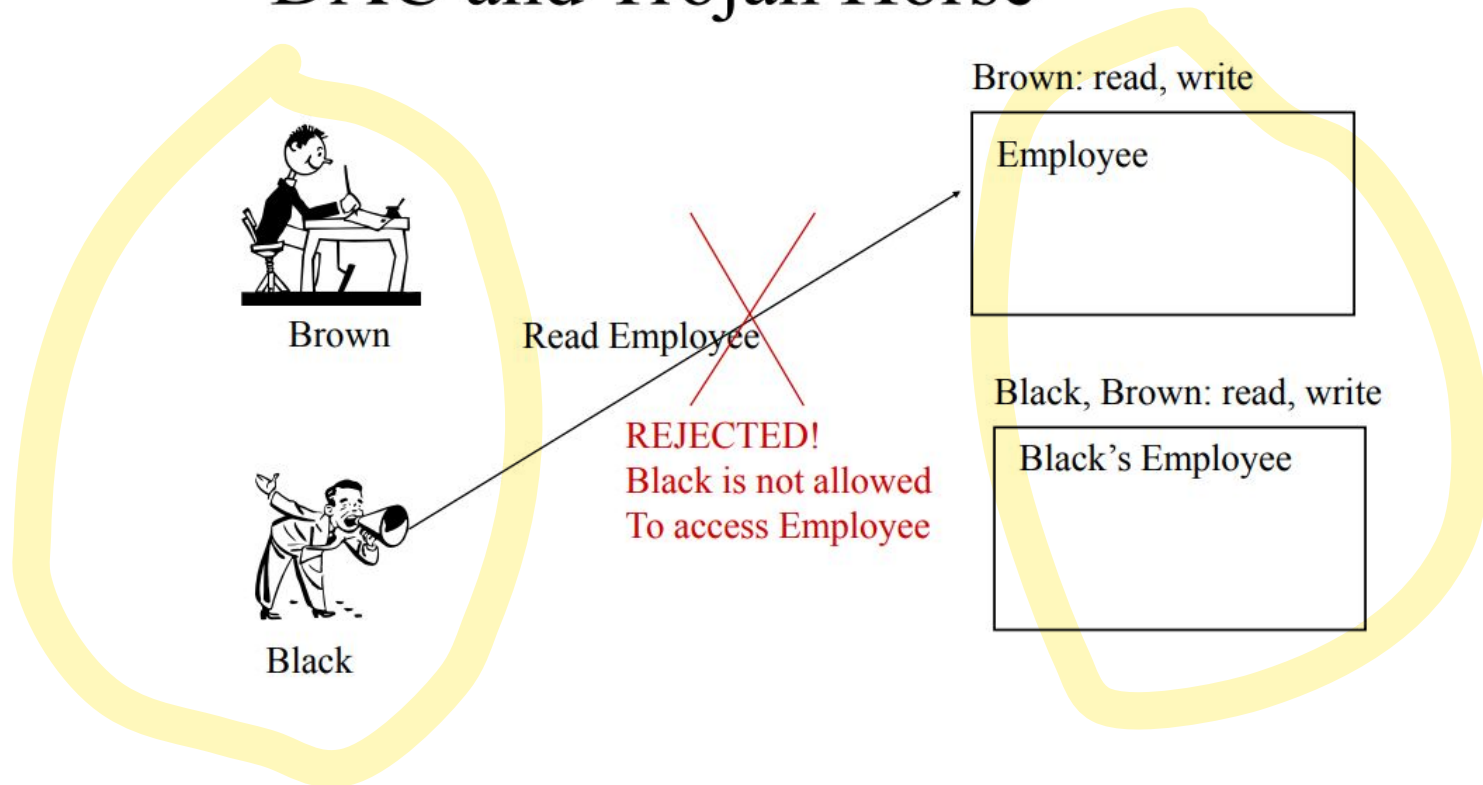
- efficient with **per subject** operations
- Usually **objects** change and **subjects** stay, so inefficient
- Capabilities are optional in POSIX (Linux and BSD).



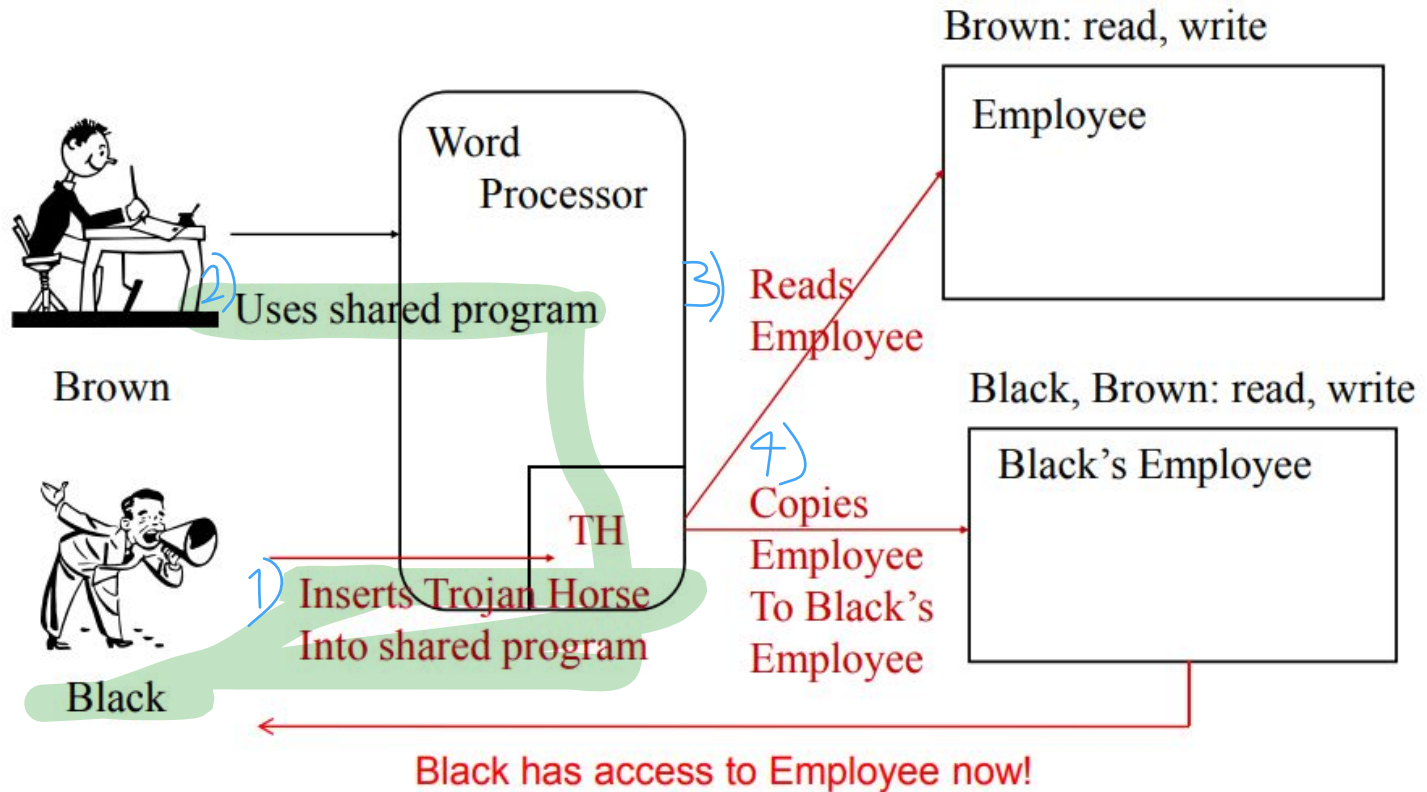
General DAC shortcomings

- Cannot prove safety
- Control access to objects but not to the data inside objects (granularity)
 - Susceptible to *malicious user* problem
 - Susceptible to *trojan horse* problem: malicious program running with privileges of the user
- Problems of scalability and management
 - each user-owner can potentially compromise security of the system with their own decisions

DAC and Trojan Horse



DAC Trojan Horse Problem



Mandatory Access Control (MAC)

Idea: do not let owners assign privileges.

Privileges are set by a security **administrator**:

- E.g., defines a **classification** of *subjects* (or "clearance") and *objects* (or "sensitivity").

The **classification** is composed of:

- A strictly ordered set of **secrecy levels**.
- A set of **labels**.



File Help

Select:

Status

Boolean

File Labeling

User Mapping

SELinux User

Translation

Network Port

Policy Module



Revert



Customized

(example of MAC rules in SELinux)

Filter

Active	Module	Description	Name
<input type="checkbox"/>	apache	Allow httpd to act as a FTP server by listening on the	httpd_enable_ftp_server
<input type="checkbox"/>	apache	Allow HTTPD to run SSI executables in the same dom	httpd_ssi_exec
<input type="checkbox"/>	apache	Allow Apache to communicate with avahi service via	allow_httpd_dbus_avahi
<input checked="" type="checkbox"/>	apache	Allow httpd to use built in scripting (usually php)	httpd_builtin_scripting
<input type="checkbox"/>	apache	Allow http daemon to send mail	httpd_can_sendmail
<input type="checkbox"/>	apache	Allow httpd to access nfs file systems	httpd_use_nfs
<input checked="" type="checkbox"/>	apache	Unify HTTPD to communicate with the terminal. Nee	httpd_tty_comm
<input type="checkbox"/>	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_ntlm_winbind
<input type="checkbox"/>	apache	Allow HTTPD scripts and modules to connect to the r	httpd_can_network_connect
<input checked="" type="checkbox"/>	apache	Unify HTTPD handling of all content files	httpd_unified
<input type="checkbox"/>	apache	Allow apache scripts to write to public content. Dire	allow_httpd_sys_script_anon_write
<input checked="" type="checkbox"/>	apache	Allow httpd to read home directories	httpd_enable_homedirs
<input checked="" type="checkbox"/>	apache	Allow Apache to modify public files used for public fi	allow_httpd_anon_write
<input type="checkbox"/>	apache	Allow Apache to use mod_auth_pam	allow_httpd_mod_auth_pam
<input type="checkbox"/>	apache	Allow httpd to access cifs file systems	httpd_use_cifs
<input checked="" type="checkbox"/>	apache	Allow httpd cgi support	httpd_enable_cgi
<input type="checkbox"/>	apache	Allow HTTPD scripts and modules to network connect	httpd_can_network_connect_db
<input type="checkbox"/>	apache	Allow httpd to act as a relay	httpd_can_network_relay
<input type="checkbox"/>	bind	Allow BIND to write the master zone files. Generally	named_write_master_zones
<input type="checkbox"/>	cdrecord	Allow cdrecord to read various content. nfs, samba, r	cdrecord_read_content
<input type="checkbox"/>	cron	Enable extra rules in the cron domain to support fcr	fcron_cron
<input type="checkbox"/>	cvs	Allow cvs daemon to read shadow	allow_cvs_read_shadow
<input checked="" type="checkbox"/>	domain	Allow unlabeled packets to work on system	allow_unlabeled_packets
<input type="checkbox"/>	exim	Allow exim to connect to databases (postgres, mysql	exim_can_connect_db
<input type="checkbox"/>	exim	Allow exim to create, read, write, and delete unprivi	exim_manage_user_files
<input type="checkbox"/>	exim	Allow exim to read unprivileged user files.	exim_read_user_files
<input type="checkbox"/>	ftp	Allow ftp to read and write files in the user home dire	ftp_home_dir
<input type="checkbox"/>	ftp	Allow ftp servers to login to local users and read/writ	allow_ftpd_full_access
<input type="checkbox"/>	ftp	Allow ftp servers to use nfs used for public file trans	allow_ftpd_use_nfs

Secrecy Levels (US example)

Top Secret

>

Secret

>

For Official Use Only (FOUO)

>

Unclassified

Secrecy Levels (NATO example)

COSMIC Top Secret

>

NATO Secret

>

NATO Confidential

>

Unclassified

Example (labels)

Policy

Energy

Finance

ATOMAL

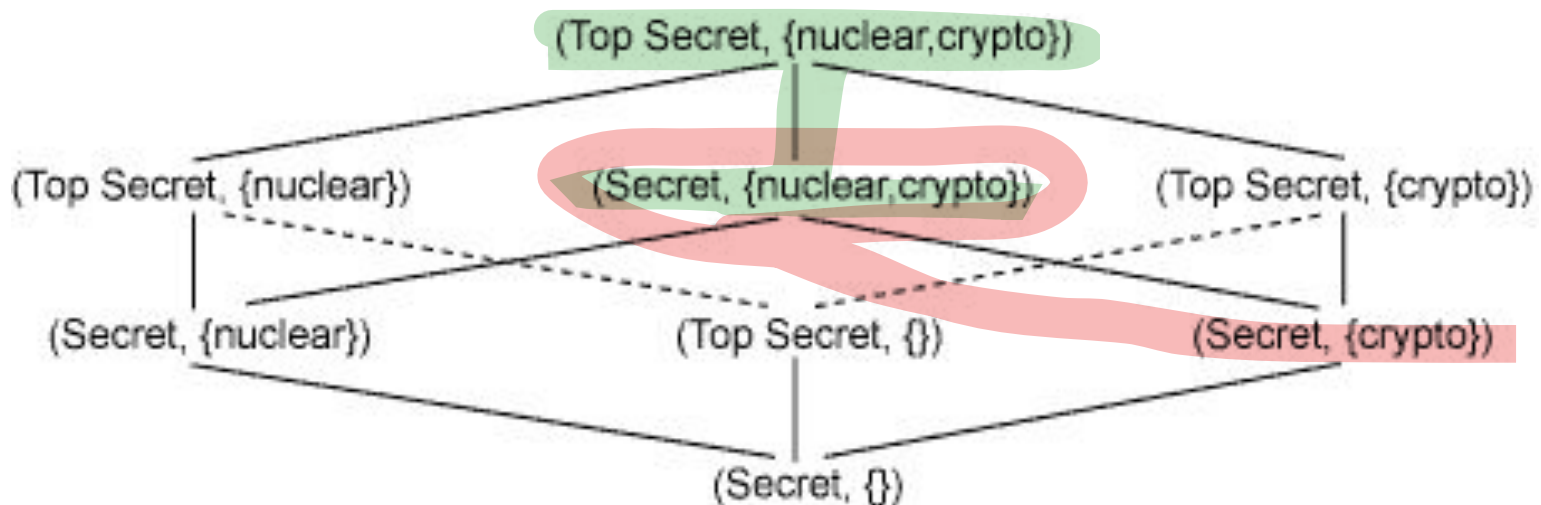
NOFORN

Secrecy Levels + Labels = Lattice (LBAC)

Classification = *partial order relationship*.

Dominance in a lattice is defined as:

$$\{C_1, L_1\} \geq \{C_2, L_2\} \Leftrightarrow C_1 \geq C_2 \text{ and } L_2 \subseteq L_1$$



(reflexive, transitive, antisymmetric property)

Bell-LaPadula Model (BLP) - 1

Defines two MAC rules:

1. **Rule 1** (no read up, "simple security property")
A subject s at a given secrecy level **cannot read** an object o at a **higher** secrecy level.
1. **Rule 2** (no write down, "star property")
A subject s at a given secrecy level **cannot write** an object o at a **lower** secrecy level.

Defines one DAC rule:

Rule 3 (Discretionary Security Property) states the use of an access matrix to specify the discretionary access control.

Bell-LaPadula Model (BLP) - 2

Tranquility property: secrecy levels of objects cannot change dynamically.

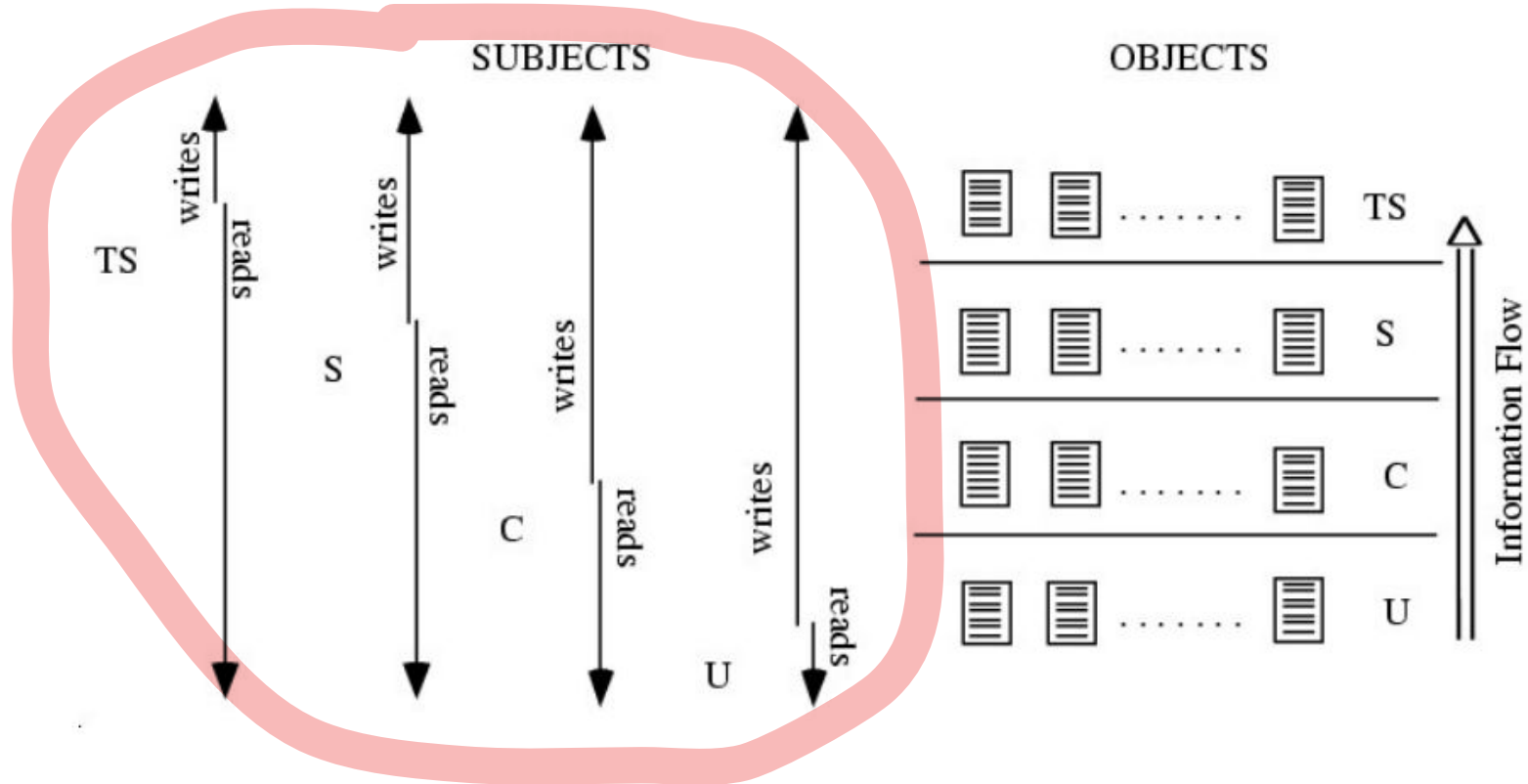
Result: monotonic flow of information toward higher secrecy levels

- need of **trusted subjects** who can declassify or sanitize documents

Limitations: does not address integrity. There are other models for integrity, e.g.

http://en.wikipedia.org/wiki/Biba_Model

MAC Information Flow



Conclusions

Access control, or authorization, defines *subjects*, *objects*, and *actions* in a system.

Access control **models** define how actions are (un)assigned to subjects and objects.

DAC are more common and “natural” than **MAC**, but can coexist.