# Access Control

Applied Information Security
Lecture 09

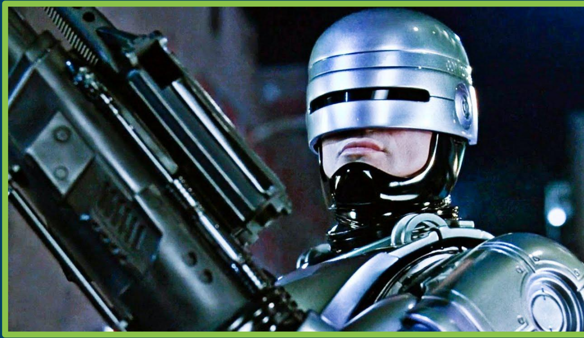
# Course So Far

**goal**: enforce policies:

“The system shall { prevent, detect } [ *action* ] { on, to, with } [ *asset* ].”

we have (in theory) seen how to handle *action*s of illegitimate users:

- **cryptography** to create a secure channel,
- **authentication** through that channel.



how do we enforce policies (confidentiality, integrity) for legitimate users?

# Gold Standard

Butler W. Lampson

**authenticate** principals

- “Who said that?”
- “Who is getting that information?”

done

**authorize** access

- “Who can do which operation on which object?”

**now**

**audit** decision of guard

- “What happened? Why?”

done

# Today’s Topics

from authentication to authorization:
access control! (AC) models:

| | |
|---|---|
| ● DAC | discretionary AC |
| ○ ACL | AC lists |
| ○ Capabilities | discretionary AC |
| ● MAC | mandatory access control |
| ○ MLS | multi-level security [Bell-Lapadua, Biba] |
| ○ commerce | [Brewer-Nash] |
| ● RBAC | role-based AC |
| ● other models | ABAC, ReBAC |

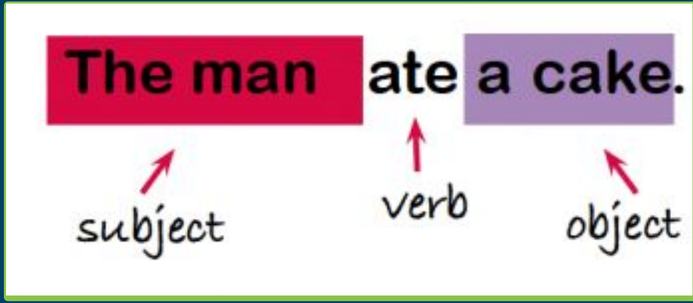
enforcement: reference monitor

# Access Control: Actors

access control *policy* specifies access rights (*permitted operations*).
regulate whether requests by principals should be permitted or denied.

principal:

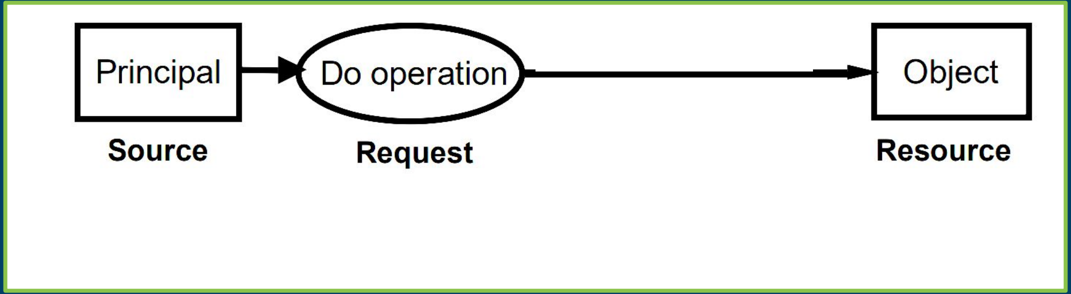
- user a human
- subject process executing on behalf of user
- object resource (e.g. piece of data)



why distinction: processes can be controlled; humans can't.

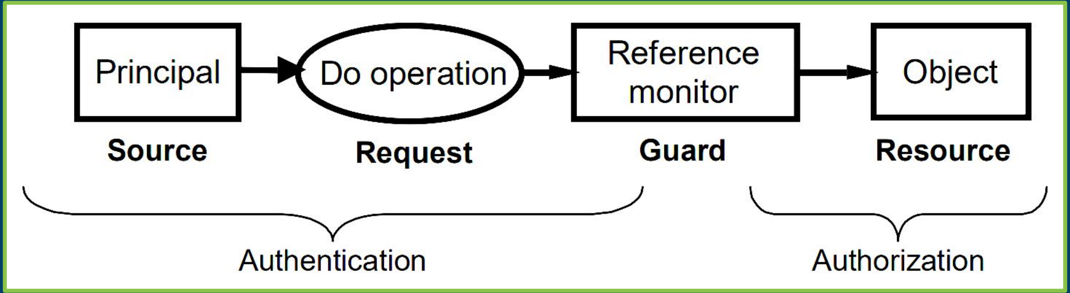
# Access Control: Model

basic model: subject (principal) attempts to access (do operation) on object.



# Access Control: Model

basic model: subject (principal) attempts to access (do operation) on object.



<u>complete mediation</u>: must check every access!

**<u>assumptions:</u>**

- *principal* can learn/update info <u>only</u> using predefined *operations*.
- operations <u>intercepted</u> by a (reference) *monitor*
- monitor only allows the operation if the principal has the required *privileges*.

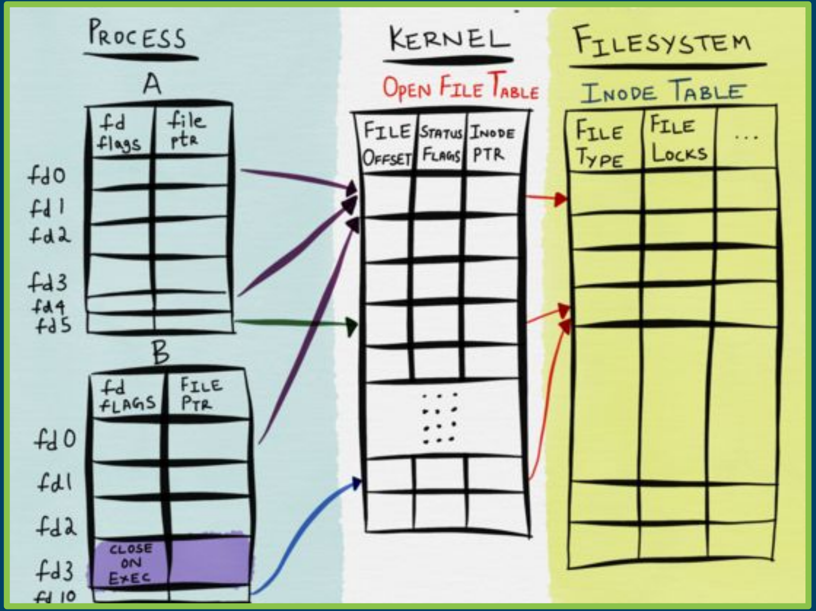
Recap (“Assurance” lecture)

# Complete Mediation

monitor and control every operation to every object by every principal.

OS reference monitor does this.
This is a form of AC!

UNIX: process tries to read a file ⇒ **kernel** decides if process is allowed.



# Access Control: Policy, Mechanism

AC policy: *principal* permitted to perform *operation* on *object*?

**confidentiality**: restrict operations that *reveal* information.

**integrity:** restrict operations that perform *updates*.

must be tamper-proof

The primary concerns of an access control mechanism:

failsafe default

- **prevent** access: ensure that subject cannot access object w/o privilege
- **determine** access: decide if subject has access (as per policy), to op on object.
- **grant** access: give a subject access to an object.
- **revoke** access: remove a subject's access to an object.
- **audit** access: determine which subjects can access object, or which objects a subject can access.

separation of privilege: don't grant access to many objects just to enable access to one.

# Access Control: Implementation Concerns

- flexibility
  - how expressive is the means of expressing policies?
- understandability
  - how complex are access control policies (depends on the flexibility)
- run-time cost
  - depends on the granularity

# Access Control: Kinds

broadly, two kinds of AC (“who controls the policy?”).

focus today

| | |
|---|---|
| discretionary | access is the discretion of the data owner |
| mandatory | an authority mandates the control |

other AC is "just" a particular way to realize DAC/MAC, and often mix DAC/MAC

- RBAC, ABAC, ...

# DAC

discretionary access control

DAC

# Discretionary Access Control

access is the discretion of the data owner

- object owner controls initial assignment of privileges on object
- object owners have the possibility of updating privileges

example: commercial OSs

- principal: user
- object: file, I/O device, ...

```
$ ls -l file
```

```
rw-rw-r--    file
```

# Policies: Access Matrix

Butler Lampson anecdote

- DAC policies can be depicted using the following table (*access matrix*):

| | object | | |
|---|---|---|---|
| principal | c1.tex | c2.tex | invtry.xls |
| fbs | r,w | r,w | r |
| mmb | | | r,w |
| jhk | | | r |

The example tables, hereafter, are from Chapter 7 of Fred B Schneider's book

# Policies: Access Matrix

- DAC policies can be depicted using the following table (*access matrix*):

| | object | | |
|---|---|---|---|
| principal | c1.tex | c2.tex | invtry.xls |
| fbs | r,w | r,w | r |
| mmb | | | r,w |
| jhk | | | r |

read operation

write operation

The example tables, hereafter, are from Chapter 7 of Fred B Schneider's book

# Access Matrix as a Relation

- set *Auth* contains triples of the form ⟨P, O, op⟩ where
    - P principal
    - O object
    - op operation

  read ⟨P, O, op⟩ ∊ Auth as “P authorized to op on O”

- *Auth* = {⟨fbs,c1.tex,r⟩, ⟨fbs,c1.tex,w⟩, ⟨fbs,c2.tex,r⟩, ⟨fbs,c2.tex, w⟩,
  ⟨fbs,invtryxls.tex,r⟩, ⟨mmb,invtryxls.tex,r⟩,
  ⟨mmb,invtryxls.tex,w⟩, ⟨jhk,invtryxls.tex,r⟩}

DAC

# Commands

- **commands** define which changes to Auth are permitted.
- example: U can authorize U’ to op on O iff U owns O.

$$
\begin{array}{l}
\textit{addPriv}(U, U', O, op): \\
\quad \underline{\text{pre}}: \textit{invoker}(U) \land \langle U, O, \texttt{owner}\rangle \in \textit{Auth} \land op \neq \texttt{owner} \\
\quad \underline{\text{action}}: \textit{Auth} := \textit{Auth} \cup \{\langle U', O, op\rangle\}
\end{array}
$$

<u>pre</u> does not hold ⇒ <u>action</u> fails.

# Protection Domain

a set of principals

ops that P needs depends on task to be performed.

P doesn't need all its privileges all the time; violates least privilege.

solution: **protection domain**.

threads -- transition between →
protection domains -- consist of →
principals

example transitions: invoke a program, change from usermode to supervisor mode, ...

# Access Matrix w/ Protection Domains

- Access matrix with protection domains

sh running on behalf of fbs

| | object | | |
|---|---|---|---|
| domain (principal) | c1.text | c2.tex | invtry.xls |
| fbs▹sh | | | |
| fbs▹gedit | r,w | r,w | |
| fbs▹excel | | | r |
| mmb▹sh | | | |
| mmb▹gedit | | | |
| mmb▹excel | | | r,w |

# Protection Domains, Transitions

OS supports protection-domains => certain sys-calls cause transition.

transitions ought to only be authorized between certain pairs of protection domains.

ex: sh ---> excel, sh ---> edit
excel -/-> sh

can specify with '`enter`' privilege, per protection domain. ⟨D, D’, `enter`⟩
(i.e. which domain can be entered from this domain)

# Protection Domain, Transitions

- Domains are principals as well as objects, the privilege e allows transition

| | object | | | | | | |
|---|---|---|---|---|---|---|---|
| domain (principal) | c1.text | c2.tex | invtry.xls | fbs▹sh | fbs▹gedit | fbs▹excel | ... |
| fbs▹sh | | | | | e | e | ... |
| fbs▹gedit | r,w | r,w | | | | | ... |
| fbs▹excel | | | r | | | | ... |
| mmb▹sh | | | | | | | ... |
| mmb▹gedit | | | | | | | ... |
| mmb▹excel | | | r,w | | | | ... |

# Confused Deputy Attack

Wormhole vulnerability in an Android lib (Baidu, …)

transitions can
reduce (i.e. attenuate)
increase (i.e. amplify)
privileges.

both needed for least privilege:

- attenuation: restricted delegate
  - P only grants P’ privileges P’ needs to do task.
- amplification: data abstraction
  - users deliberately kept ignorant of how O is implemented.

**confused deputy attack**: P deputy of A, doing things for A that P is privileged to do but A is not.

DAC - Confused Deputy Attack

# Example

- Client may issue a request to abuse the privileges of a server

```
Server: remoteExec(File f, Operation op)
        buffer  := System.read(f)
        results := System.exec(buffer, op)
        charges := calculateBill(results)
        System.write(f, results)
        System.write(charges.txt, charges)
```

DAC - Confused Deputy Attack

# Confused Deputy

- Client may issue a request to abuse the privileges of a server

If f = charges.txt  the client may modified a file she wasn’t supposed to. For instance, if Server holds the privilege to write on charges.txt but the client doesn’t.

```
Server: remoteExec(File f, Operation op)
        buffer  := System.read(f)
        results := System.exec(buffer, op)
        charges := calculateBill(results)
        System.write(f, results)
        System.write(charges.txt, charges)
```

DAC - Confused Deputy Attack

# Example, Solution?

- Check if client has permission, per statement.

```
Server: remoteExec(File f, Operation op)
        1: buffer  := System.read(f)
        2: results := System.exec(buffer, op)
        3: charges := calculateBill(results)
        4: System.write(f, results)
        5: System.write(charges.txt, charges)
```

- only instruction 5 can modify charges.txt.
- impractical; would need to be indicated in each statement in code.

DAC - Confused Deputy Attack

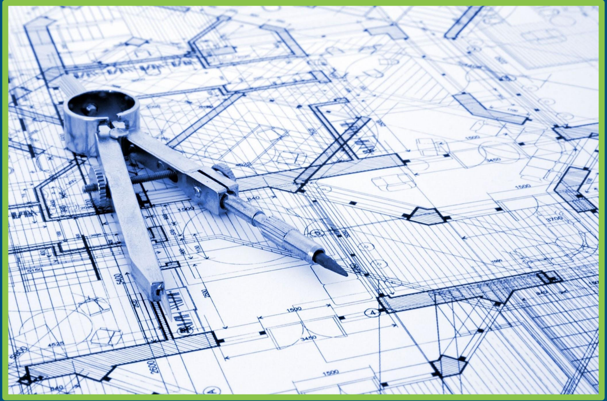
# Example, Solution

- Bundle client permissions together with object name

```
Server: remoteExec(File f, Operation op, Privs privs)
           buffer  := System.read(f)
           results := System.exec(buffer, op, privs)
           charges := calculateBill(results)
           System.write(f, results)
           System.write(charges.txt, charges)
```

Bundle together the client permissions needed to perform the operation. The server can then check that the client doesn’t have writing privileges in charges.txt.

# Implementing DAC



necessary:

- represent Auth somehow
- check ⟨P, O, op⟩ ∈ Auth (i.e. P authorized to op on O?)
- change Auth w/ commands defined by DAC policy
- associate protection domain w/ each thread of control
- have threads transition between protection domains

useful:

- given P, list all ⟨O, op⟩ s.t. ⟨P, O, op⟩∈Auth
- given O, list all ⟨P, op⟩ s.t. ⟨P, O, op⟩∈Auth

DAC - Implementing DAC

# Naive approach

store access matrix in memory

huge, mostly empty, table.

| | object | | | | | | |
|---|---|---|---|---|---|---|---|
| domain (principal) | c1.text | c2.tex | invtry.xls | fbs▷sh | fbs▷gedit | fbs▷excel | ... |
| fbs▷sh | | | | | e | e | ... |
| fbs▷gedit | r,w | r,w | | | | | ... |
| fbs▷excel | | | r | | | | ... |
| mmb▷sh | | | | | | | ... |
| mmb▷gedit | | | | | | | ... |
| mmb▷excel | | | r,w | | | | ... |

DAC - Implementing DAC

# Better Approaches

## Access Control Lists

non-empty cells associated with the **column** (object).
stored w/
the object

## Capabilities

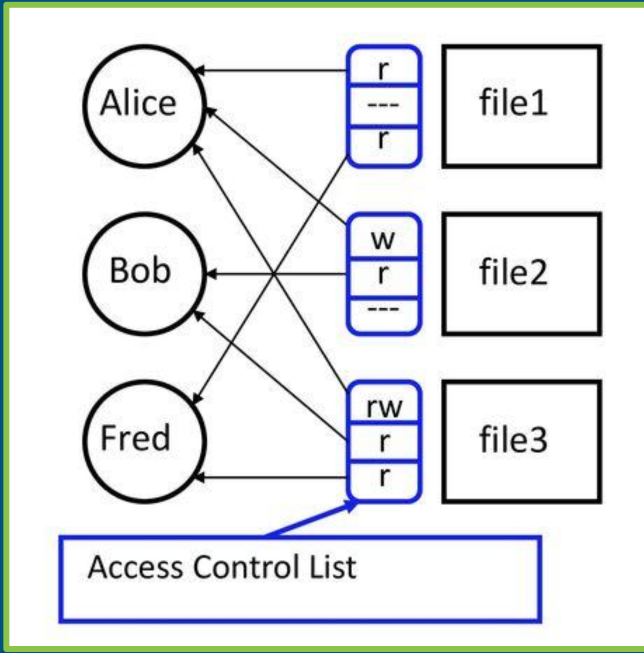
Non-empty cells associated with the **row** (principal).
stored w/ the
principal

| | object | | | | | | |
|---|---|---|---|---|---|---|---|
| domain (principal) | c1.text | c2.tex | invtry.xls | fbs▹sh | fbs▹gedit | fbs▹excel | ... |
| fbs▹sh | | | | | e | e | ... |
| fbs▹gedit | r,w | r,w | | | | | ... |
| fbs▹excel | | | r | | | | ... |
| mmb▹sh | | | | | | | ... |
| mmb▹gedit | | | | | | | ... |
| mmb▹excel | | | r,w | | | | ... |

DAC

# ACL

## access control list



DAC - ACL

# Access Control Lists (Acls)

- the ACL of an O is a list

$$\langle P_1, Privs_1\rangle, \langle P_2, Privs_2\rangle, \ldots$$

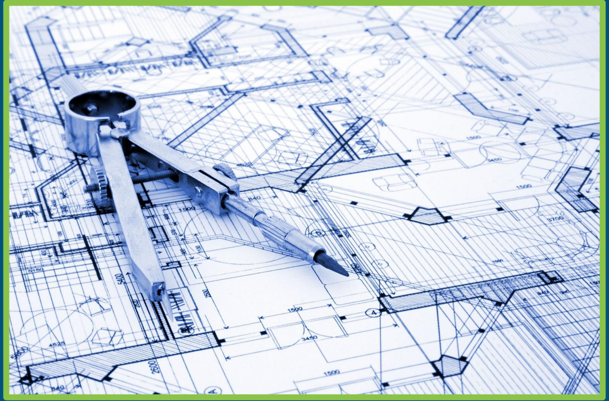
- an *ACL entry* $\langle P_i, Privs_i\rangle$ is in the list for an object *O* iff

$$\forall\ op \in Privs_i\ .\ \langle p_i, O, op\rangle \in Auth$$

- example

$$\langle \mathtt{fbs}, \{\mathtt{r}\}\rangle, \langle \mathtt{mmb}, \{\mathtt{r}, \mathtt{w}\}\rangle, \langle \mathtt{jhk}, \{\mathtt{r}\}\rangle$$

# Implementing ACL



long lists are unwieldy. Simplify:

**groups of principals:** access is granted by virtue of being in group.

- P gets access to course notes by virtue of being an enrolled student.
- P gets access to the ITU HPC by virtue of being faculty.

encoding groups in ACL, problems:

- add/rem. U requires upd. many ACLs.
- rem. P from G by rem. P from ACL of O: but P could access O by other means...

solution: group declaration ⟨G,Privs⟩

represents set of principals

DAC - ACL - Implementing ACL

# Negative Policies?

to conclude that P cannot op on O, must check all P granted op by the ACL of O. time-consuming (especially when groups are present).

solution (?): prohibitions (e.g. negative policies, !op).

- ⟨P,{!op}⟩ in ACL of O means P cannot op on O
- warning: conflicting policies; what if P can both op and !op on O?
  (take first privilege found in ACL?)

**anecdote** (a real issue): Facebook is working on reducing the ACL checking time.

# Where to keep the ACL?

ACL must be stored s.t. its integrity is protected.

**solutions**:

monitor

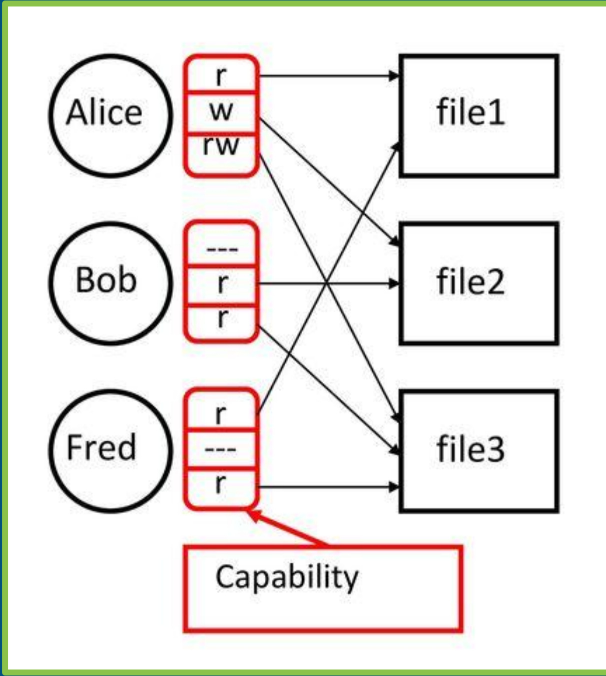
- store the ACL w/ O, so updates to the ACL are checked by M.
- store the ACL w/ M, so integrity-protection of M protects integrity of ACL.

OS abstractions are a good place to store the ACL.

- files: big enough to house ACL.
- locks/ports: few; easy to keep in OS memory.
- M is part of the OS.

# Capabilities

## capability-based security



# Capabilities

a capability is a pair ⟨O,Privs⟩.
a principal can hold a capability (is granted Privs on O).

*Auth* is enforced, provided the following holds throughout execution:

1. for P to op on O, P must hold ⟨O,Privs⟩ with op ∈ Privs.
   *capability-based addressing* *W: possession of capability*
2. ⟨O,Privs⟩s cannot be counterfeited or corrupted.
   *capability authenticity* *W: integrity of capability*

# Possession of Capability

W: “*possession of capability*”

⟨O,Privs⟩s are the sole means by which P identify & access O.

solves confused deputy.

privilege bundled w/ file name

# Capability Authenticity

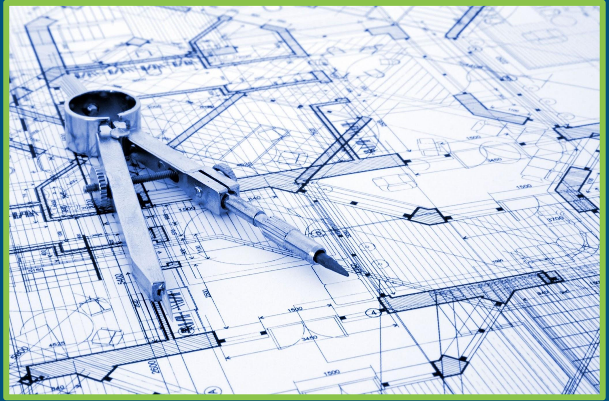
W: *"integrity of capability"*

prevent unauthorized creation / changes of ⟨O,Privs⟩s.

implementation approaches:

- tagged memory (HW)
- protected address space (HW)
- cryptographically-protected
- type-safety-protected

# Implementing Capabilities



P can

1. create new O, and in so doing, receive ⟨O,Privs⟩
2. transfer to P' one or more capability P holds (w/ attenuation/amplification)
3. revoke capabilities that derive from ⟨O,Privs⟩.

all capabilities are derived from the capability received initially by *creator* (i.e. owner) of O.

privileges controlled by owner (or principals whose authority can be traced to the owner). Thus DAC.

DAC

# ACL vs. Capabilities

## ACL

- implemented as reference-monitor + list.
  - localized
- attractive; separation-of-concerns.
- but: high cost in managing the many small protection domains.
- must be defined explicitly for instantiating Least Privilege.

## Capabilities

- this complexity is eliminated; protection domains are not enforced explicitly.
- but, w/ capabilities: to discover what accesses are, or could become, possible as execution proceeds, large set of components must be analyzed.
- but, this decentralization is exactly what makes capabilities appealing for controlling access to user-defined objects.

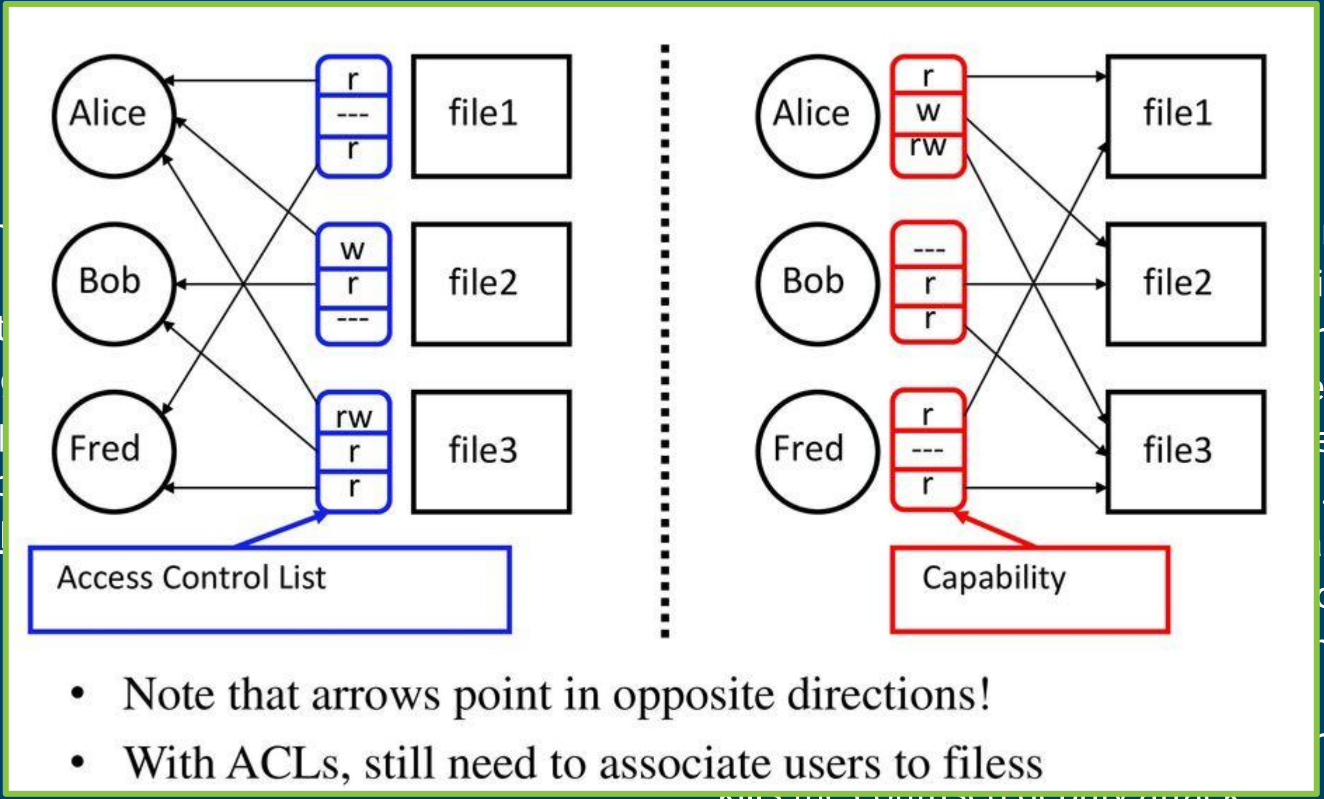
Capabilities combine naming+authorization.
kills the confused deputy attack.

DAC

# ACL vs. Capabilities

## ACL

- implen
  - l
- attract
- but: hi
  small
- must b
  Least



- Note that arrows point in opposite directions!
- With ACLs, still need to associate users to filess

rotection
icitly.
what
, possible
et of
.
ctly what
or
ned objects.

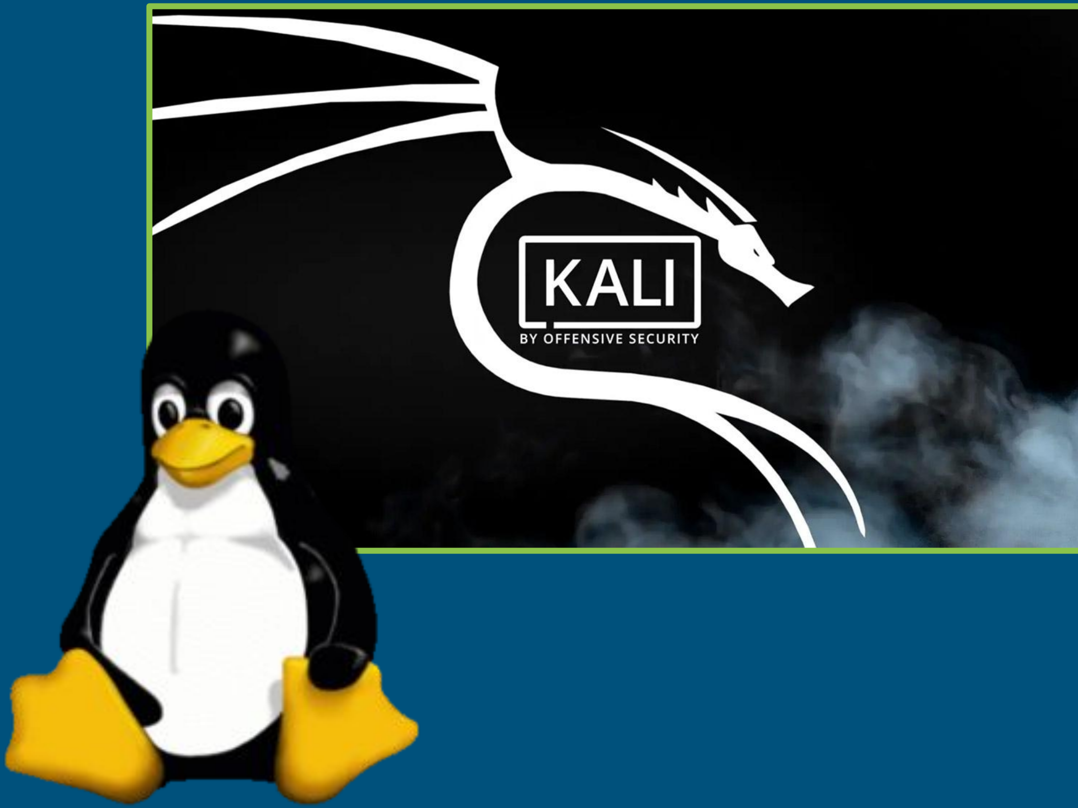
rization.

kills the confused deputy attack.

DAC

# UNIX

## case study



# Case Study: UNIX

- authorize requests that processes make to perform operations on files
  - file descriptors are used for: files, devices, sockets, …
- principals
  - users
  - groups of users
- objects
  - files (“everything is a file”)
- each file has
  - an ACL associated to it,
  - a user id (the owner), and
  - a group id (file’s group)

DAC - UNIX

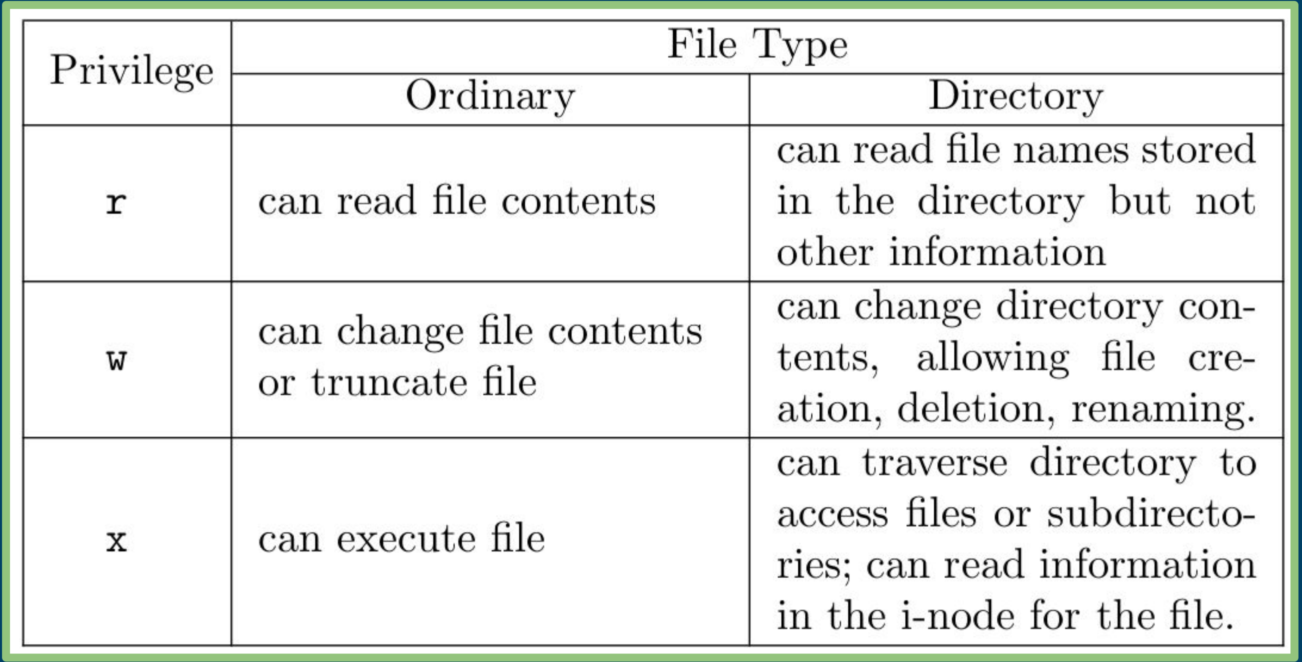
# Case Study: UNIX

- sacrifices expressiveness in favor of succinctness
  - each file defines a set of privileges for its owner, group and other users

$$\text{Privs}_F.\text{owner} = \{\texttt{r}, \texttt{w}\} \quad \text{Privs}_F.\text{group} = \{\texttt{r}\} \quad \text{Privs}_F.\text{others} = \varnothing$$

DAC - UNIX

# UNIX Privileges Interpretation



| Privilege | File Type | |
|---|---|---|
| | Ordinary | Directory |
| r | can read file contents | can read file names stored in the directory but not other information |
| w | can change file contents or truncate file | can change directory contents, allowing file creation, deletion, renaming. |
| x | can execute file | can traverse directory to access files or subdirectories; can read information in the i-node for the file. |

# UNIX: Authorization Check

- process w/ user id, *euid*, and group id, *egid*, is authorized to perform an operation requiring a privilege *p* iff

$$(p \in \text{Privs}_F.\text{owner} \ \wedge \ \text{euid} = \text{owner}_F)$$

need to be owner, am owner

$$\vee \ (p \in \text{Privs}_F.\text{group} \ \wedge \ \text{euid} \neq \text{owner}_F \ \wedge \ \text{egid} = \text{owner}_F)$$

$$\vee \ (p \in \text{Privs}_F.\text{others} \ \wedge \ \text{euid} \neq \text{owner}_F \ \wedge \ \text{egid} \neq \text{owner}_F)$$

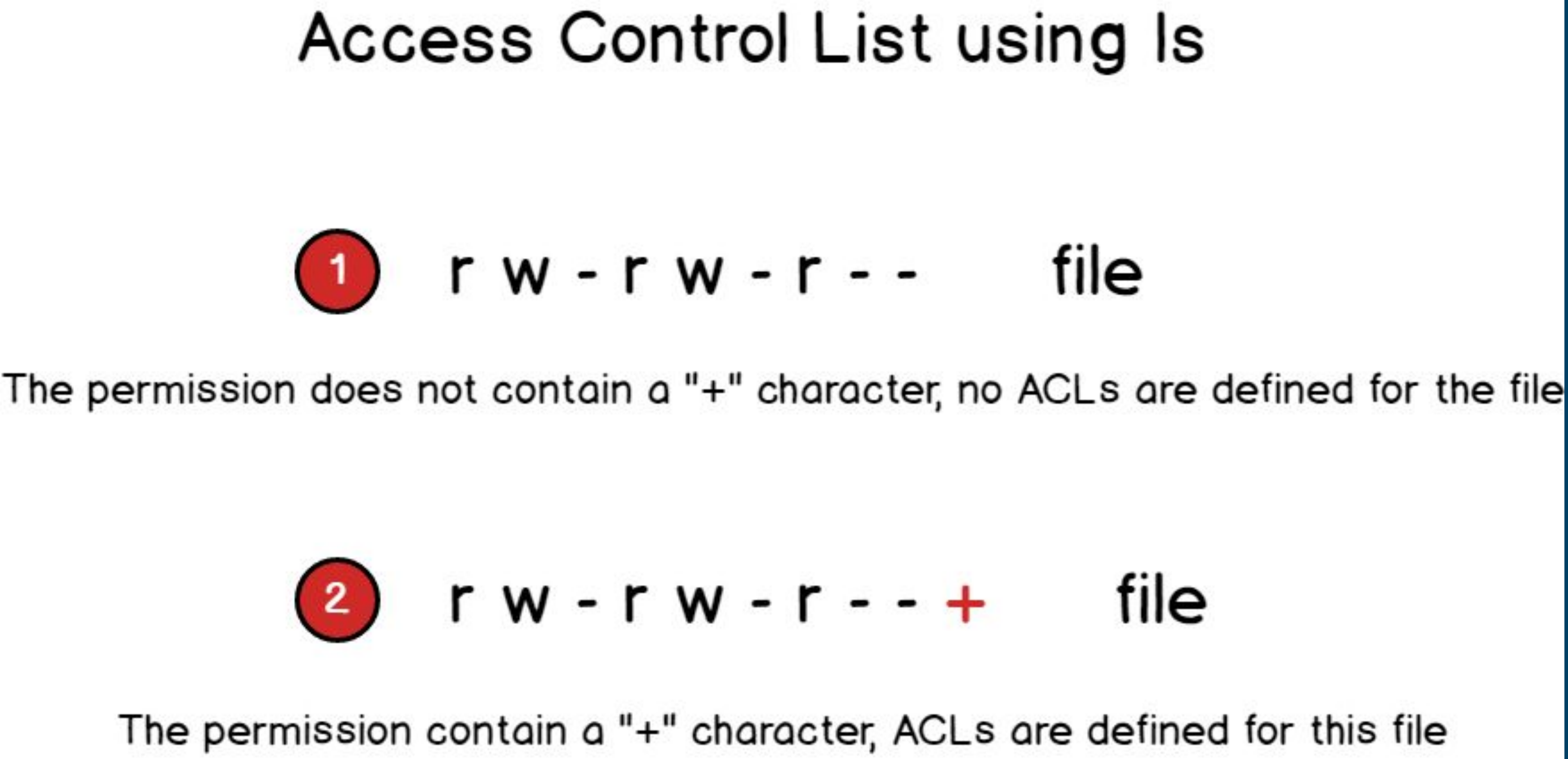
# UNIX: Domain Transitions

- `suid`
  - when a file *F* having `suid` executes, it causes a change of user id to $\mathit{owner}_F$
  - used by system services
  - `root` uid required for manipulating restricted parts of the OS

```
-rwsr-xr-x 1 root root 40760  Sep 26  2013 /bin/ping
-rwsr-xr-x 1 root root 77336  Apr 28  2014 /bin/mount
-rwsr-xr-x 1 root root 30768  Feb 22  2012 /usr/bin/passwd
---s--x--x 1 root root 123832 Nov 22  2013 /usr/bin/sudo         etc.
```

- `sgid`
  - When a file *F* having `sgid` executes, it causes a change of the group id to $\mathit{group}_F$

# DAC - UNIX



## Access Control List using ls

1 rw-rw-r-- file

The permission does not contain a "+" character, no ACLs are defined for the file

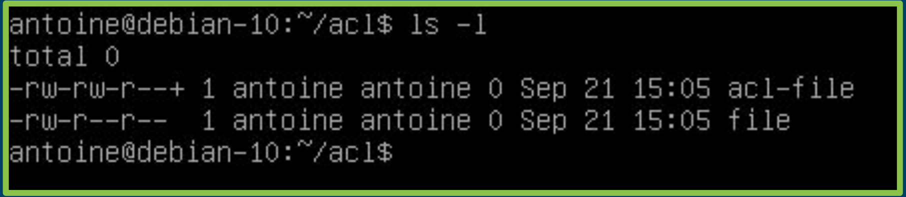
2 rw-rw-r--+ file

The permission contain a "+" character, ACLs are defined for this file

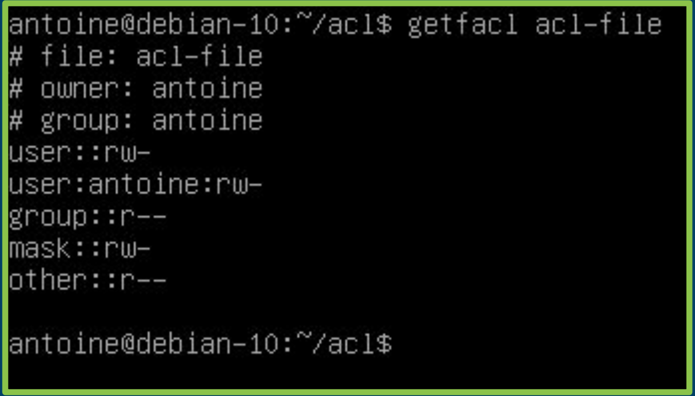
https://devconnected.com/access-control-lists-on-linux-explained/

DAC - UNIX

# ACL in Linux



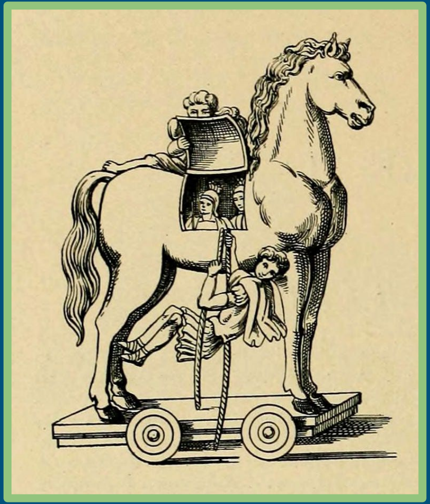
```
antoine@debian-10:~/acl$ ls -l
total 0
-rw-rw-r--+ 1 antoine antoine 0 Sep 21 15:05 acl-file
-rw-r--r--  1 antoine antoine 0 Sep 21 15:05 file
antoine@debian-10:~/acl$
```



```
antoine@debian-10:~/acl$ getfacl acl-file
# file: acl-file
# owner: antoine
# group: antoine
user::rw-
user:antoine:rw-
group::r--
mask::rw-
other::r--

antoine@debian-10:~/acl$
```

# Trojan Horse Attack



- DAC vulnerable Trojan horse attacks.
- consider the following scenario
  1. user runs a program with access to confidential information
  2. the program creates or chooses a file with reading permission for the attacker
  3. the program reads confidential information
  4. the program writes the confidential information in the file that it is readable for the attacker
- (think Android; need file sys for notes, internet for ads ⇒ can dump file sys on the Web)

crux of the problem: once P has *access*, P can do anything he/she wants with that access.

- stronger AC models prevent this.

DAC

# Summary

**policies**: "The system shall { prevent, detect } [ *action* ] { on, to, with } [ *asset* ]."

**enforce**: *authenticate, authorize, audit* (gold standard)

- secure communication over untrusted medium
- authenticate users
- authorize access to information
- audit decision of guard

open ends:

- what about that Trojan horse? (untrustworthy SW)

# MAC

mandatory access control

# 1. MULTI-LEVEL SECURITY

# Sensitivity

- Concern is **confidentiality** of information
- Documents classified according to sensitivity: risk associated with release of information
- In US:
  - Top Secret
  - Secret
  - Confidential
  - Unclassified

# Compartments

- Documents classified according to compartment(s): categories of information (in fact, aka category)
  - cryptography
  - nuclear
  - biological
  - reconnaissance
- **Need to Know Principle:** access should be granted only when necessary to perform assigned duties (instance of Least Privilege)
  - {crypto,nuclear}: must need to know about **both** to access
  - {}: no particular compartments

# Labels

- Label: pair of sensitivity level and set of compartments, e.g.,
  - (Top Secret, {crypto, nuclear})
  - (Unclassified, {})
- Users are labeled according to their clearance
- Document is labeled aka classified
  - Perhaps each paragraph labeled
  - Label of document is most restrictive label for any paragraph
- Labels are imposed by organization
- **Notation:** let L(X) be the label of entity X

# Restrictiveness of labels

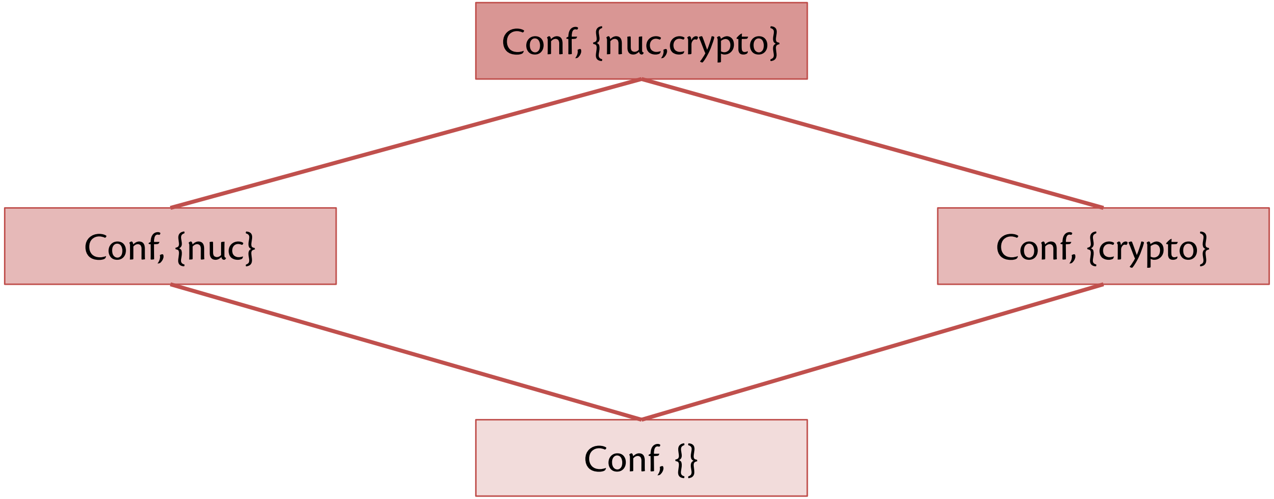
**Notation:** L1 ⊑ L2

- means L1 is no more restrictive than L2
  - less precisely: L1 is less restrictive than L2
  - another reading: information may flow from L1 to L2
  - also: L1 is dominated by L2
- e.g.
  - (Unclassified,{}) ⊑ (Top Secret, {})
  - (Top Secret, {crypto}) ⊑ (Top Secret, {crypto,nuclear})

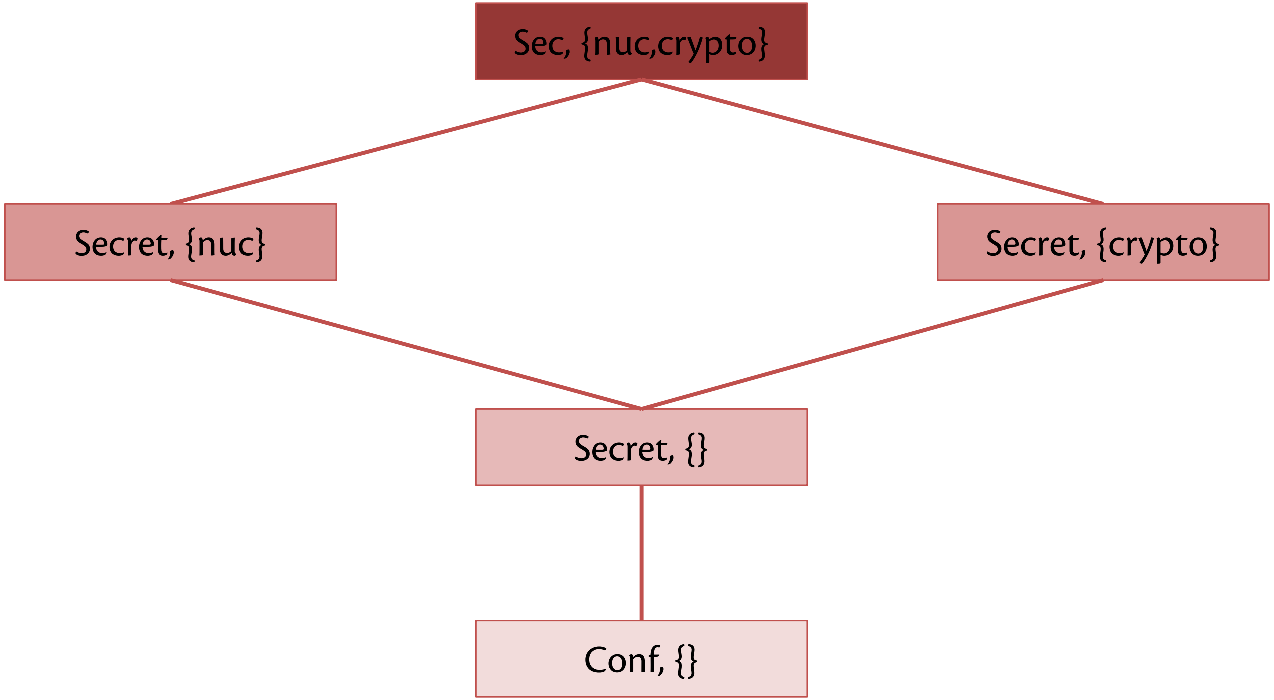
# Restrictiveness of labels

- **Definition:**
  - Let L1 = (S1, C1) and L2 = (S2, C2)
  - $L1 \sqsubseteq L2$ iff $S1 \leq S2$ and $C1 \subseteq C2$
  - Where ≤ is order on sensitivity:
    Unclassified ≤ Confidential ≤ Secret ≤ Top Secret
- Partial order:
  - Some labels are incomparable
  - e.g. (Secret, {crypto}) vs. (Top Secret, {nuclear})

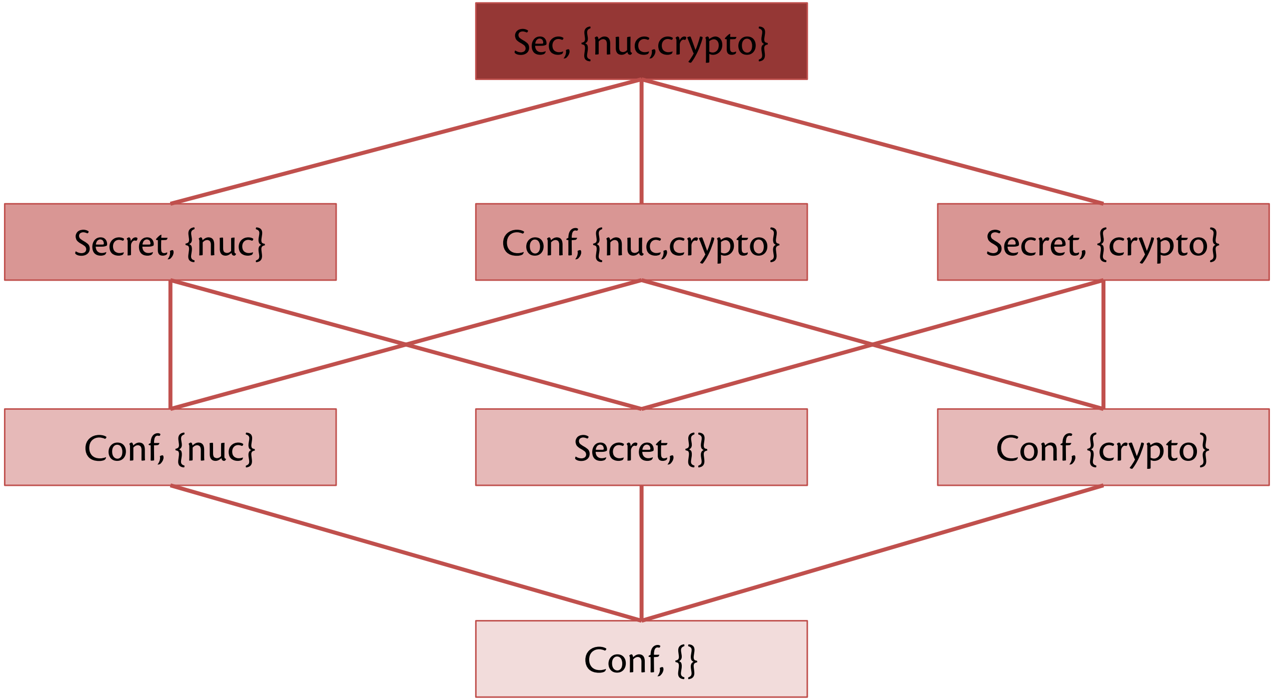
# Label partial order



# Label partial order



# Label partial order


Sec, {nuc,crypto}

Secret, {nuc}

Conf, {nuc,crypto}

Secret, {crypto}

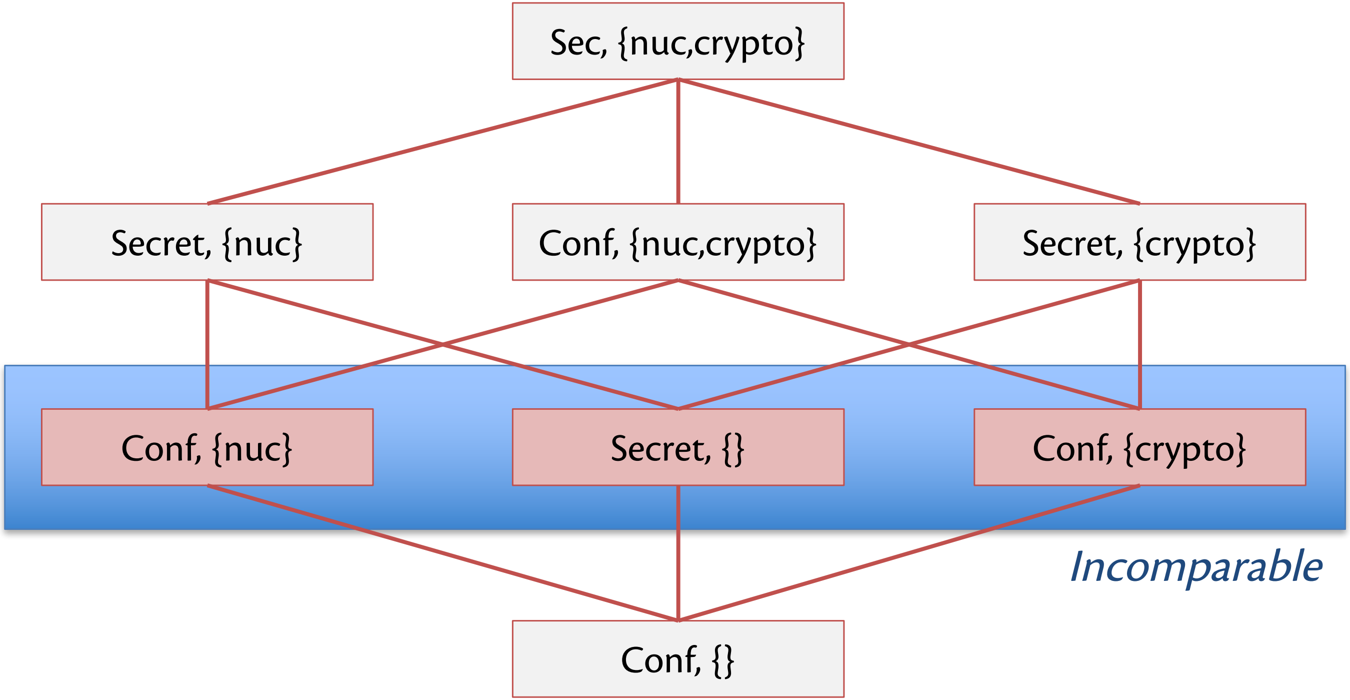
Conf, {nuc}

Secret, {}

Conf, {crypto}

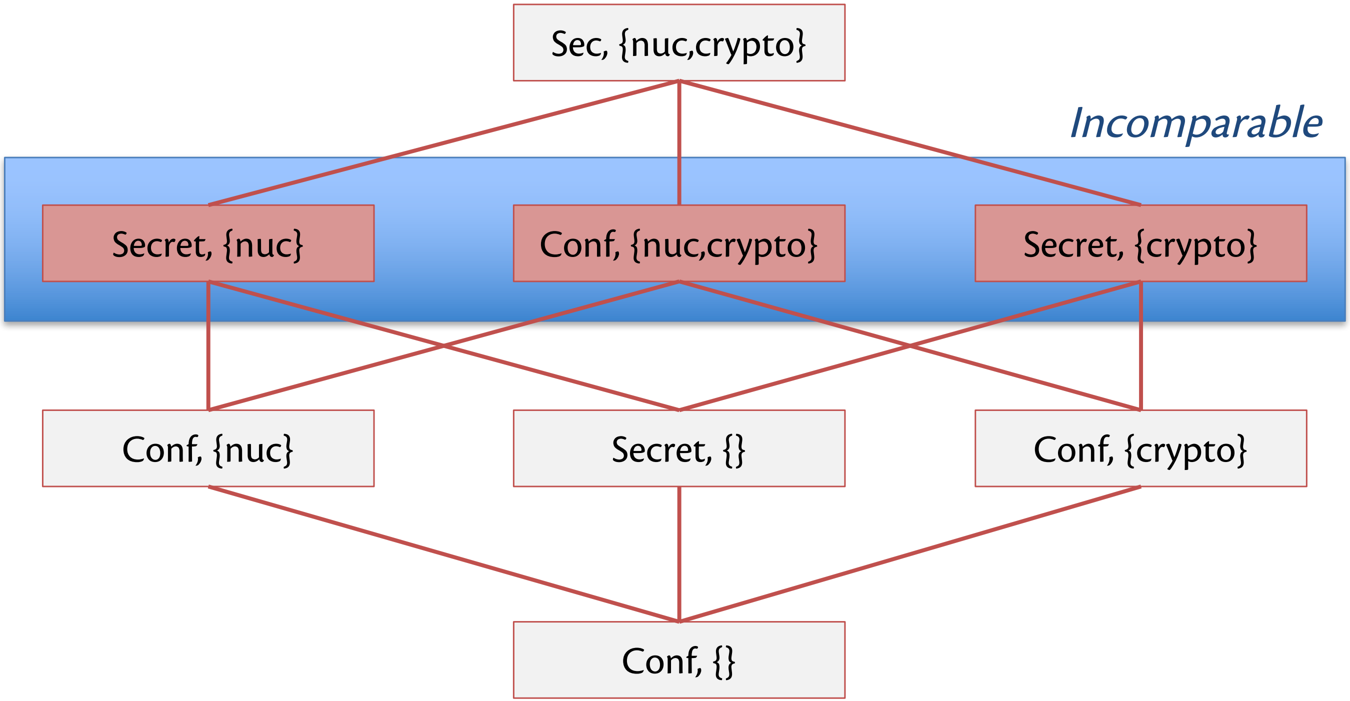
Conf, {}

# Label partial order



*Incomparable*

# Label partial order



# Access control with MLS

- When may a subject read an object?
  - **Threat:** subject attempts to read information for which it is not cleared
  - e.g., subject with clearance Unclassified attempts to read Top Secret information
- When may a subject write an object?
  - **Threat:** subject attempts to *launder* information by writing into a lower-security object
  - e.g., subject with clearance Top Secret reads Top Secret information then writes it into an Unclassified file

# Access control with MLS

Threat of concern is **subject** not **user**:

- Users trustworthy by virtue of vetting process for security clearance
- Out of scope (e.g.): user who views Top Secret information and calls the *Washington Post*
- But still want to enforce Least Privilege
- And malicious programs are a threat...

# Trojan Horse

# Access control with MLS

- When may a subject read an object?
  - **S may read O iff $L(O) \sqsubseteq L(S)$**
  - object's classification must be below (or equal to) subject's clearance
  - "no read up"
- When may a subject write an object?
  - **S may write O iff $L(S) \sqsubseteq L(O)$**
  - object's classification must be above (or equal to) subject's clearance
  - "no write down"
- Beautiful **symmetry** between these

# Reading with MLS

- Scenario:
  - Colonel with clearance (Secret, {nuclear, Europe})
  - DocA with classification (Confidential, {nuclear})
  - DocB with classification (Secret, {Europe, US})
  - DocC with classification (Top Secret, {nuclear, Europe})
- Which documents may Colonel **read**?
  - Recall: S may read O iff $L(O) \sqsubseteq L(S)$
  - DocA: (Confidential, {nuclear}) $\sqsubseteq$ (Secret, {nuclear, Europe})
  - DocB: (Secret, {Europe, US}) $\not\sqsubseteq$ (Secret, {nuclear, Europe})
  - DocC: (Top Secret, {nuclear, Europe}) $\not\sqsubseteq$ (Secret, {nuclear, Europe})

# Writing with MLS

- Scenario:
  - Colonel with clearance (Secret, {nuclear, Europe})
  - DocA with classification (Confidential, {nuclear})
  - DocB with classification (Secret, {Europe, US})
  - DocC with classification (Top Secret, {nuclear, Europe})
- Which documents may Colonel **write**?
  - Recall: S may write O iff $L(S) \sqsubseteq L(O)$
  - DocA: (Secret, {nuclear, Europe}) $\not\sqsubseteq$ (Confidential, {nuclear})
  - DocB: (Secret, {nuclear, Europe}) $\not\sqsubseteq$ (Secret, {Europe, US})
  - DocC: (Secret, {nuclear, Europe}) $\sqsubseteq$ (Top Secret, {nuclear, Europe})

# Reading and writing with MLS

- Scenario:
  - Colonel with clearance (Secret, {nuclear, Europe})
  - DocA with classification (Confidential, {nuclear})
  - DocB with classification (Secret, {Europe, US})
  - DocC with classification (Top Secret, {nuclear, Europe})
- Summary:
  - DocA: Colonel may read but not write
  - DocB: Colonel may neither read nor write
  - DocC: Colonel may write but not read

# Perplexities of writing with MLS

1. Blind write: subject may not read higher-security object yet may write it
   - Useful for logging
   - Some implementations prohibit writing up as well as writing down
2. **User** who wants to write lower-security object may not
   - **Attenuation of privilege:** login at a lower security level than clearance
   - Motivated by Trojan Horse
   - Nice (annoying?) application of Least Privilege
3. Declassification violates "no write down"
   - Encryption or billing procedure produces (e.g.) Unclassified output from Secret information
   - Traditional solution is trusted subjects who are not constrained by access control rules

# Prevention of laundering

- Earlier concern: "subject with clearance Top Secret reads Top Secret information then writes it into an Unclassified file"
- More generally:
  - S reads O1 then writes O2
  - where $L(O2) \sqsubset L(O1)$
  - and regardless of $L(S)$
- **Prohibited by MLS rules:**
  - S read O1, so $L(O1) \sqsubseteq L(S)$
  - S wrote O2, so $L(S) \sqsubseteq L(O2)$
  - So $L(O1) \sqsubseteq L(S) \sqsubseteq L(O2)$
  - Hence $L(O1) \sqsubseteq L(O2)$
  - But combined with $L(O2) \sqsubset L(O1)$, we have $L(O1) \sqsubset L(O1)$
  - Contradiction!
- So access control rules would defeat laundering, Trojan Horse, etc.

# BLP

[Bell and LaPadula 1973]

- Formal mathematical model of MLS plus access control matrix
- Proof that information cannot leak to subjects not cleared for it
- "No read up": simple security property
- "No write down": *-property
- *"The influence of [BLP] permeates all policy modeling in computer security"* –Matt Bishop
  - Influenced Orange Book
  - Led to research field "foundations of computer security”

# BLP, for integrity

- BLP is about confidentiality
- Adapted to integrity by Biba [1977]: same rules, different lattice
  - Instead of Unclassified and Secret, labels could be Untrusted and Trusted
- Recall L1 ⊑ L2 means “L1 may flow to L2”
  - BLP: low secrecy sources may flow to high secrecy sinks
    - Hence Unclassified ⊑ Secret, but not v.v.
  - Biba: low integrity sources may not flow to high integrity sinks
    - Hence Trusted ⊑ Untrusted, but not v.v.
  - High vs. low is “flipped” (lattices are *duals*)

# Biba model

- **S may read O iff L(O) $\sqsubseteq$ L(S)**
  - E.g., Trusted subject cannot read Untrusted object
  - But Untrusted subject may read Trusted object
- **S may write O iff L(S) $\sqsubseteq$ L(O)**
  - E.g., Trusted subject may write Untrusted object
  - But Untrusted subject may not write Trusted object

# MLS/BLP in OSs

- SELinux [open source release by NSA 2000]
- TrustedBSD [2000], influences iOS and OS X

# 2. BREWER-NASH

# Conflict of interest

**Setting: consulting firm**

- e.g., stock exchange, investment bank, law firm
- Consultant represents two clients
  - Best interest of those clients conflict
  - Consultant could help one at expense of the other
  - Consultant has a conflict of interest (COI)
- Norms (laws, regulations, ethics) prohibit consultant from exploiting COI
- After some time (days, years, never), COI might expire

# Conflict of interest

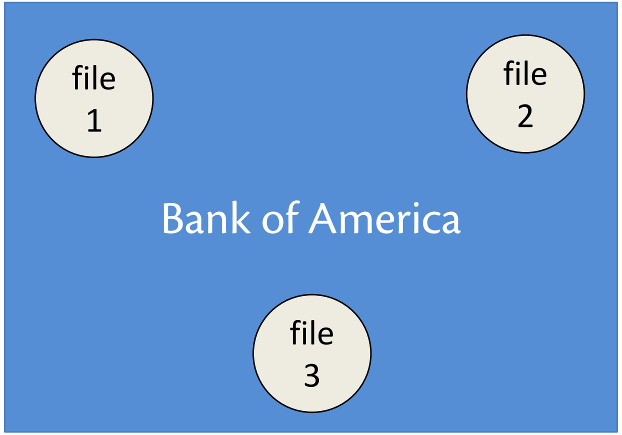
- Typical paper implementation:
  - Consultant maintains public CV
    - Entry in CV for each client
    - Entry has been sanitized and approved by client, e.g., "Sep 2015-Apr 2016: consulted on security requirements for a new branch accounting system for a major US retail bank"
  - Manager checks CV before assigning consultant to new client
  - Client receives CV to double-check from their perspective
- Brewer and Nash [1989] invented a MAC policy for this setting
  - Often known as Chinese Wall (CW)
  - Other names:  ethics wall, screen

# Great Wall of China

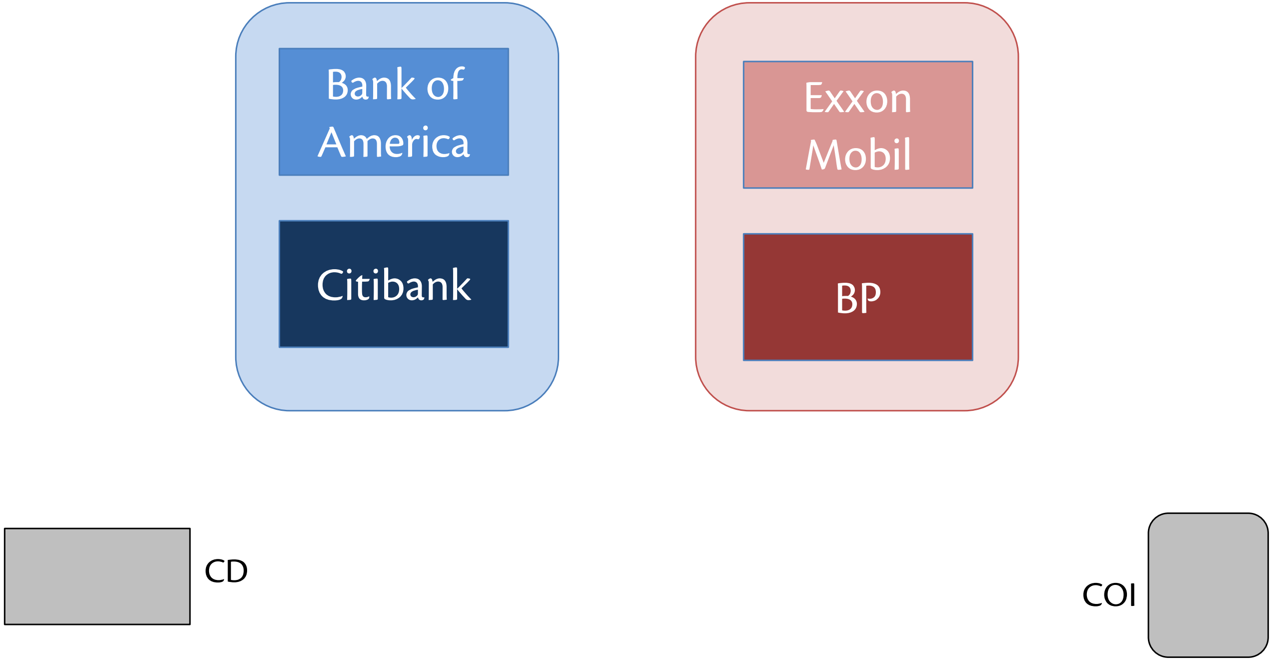
# Brewer-Nash model

- Object: contains sensitive information about companies
  - a file about Bank of America's trade secrets
  - but not its addresses, phone numbers, etc.
- Company dataset (CD): all the objects related to a single company
  - all the files about Bank of America
- Conflict of interest class (COI): all the company datasets for which the companies compete
  - all the files about banks

# Brewer-Nash model



# Brewer-Nash model



Bank of America

Citibank

Exxon Mobil

BP

CD

COI

# Breaches

**Prevent** two kinds of breaches of the wall:

- One consultant works on more than one CD inside a COI
- Two consultants each work on their own CD inside COI but cooperate to write that information to a shared object

# Access control with Brewer-Nash

- When may a subject read an object?
  - **S may read O iff**
    **S has never read any O' such that**
    **COI(O) = COI(O') and CD(O) != CD(O')**
  - Subject may not read from two CDs inside same COI
  - Requires tracking history of objects read by subject
- When may a subject write an object?
  - **S may write O iff**
    **S has never read any O' such that**
    **CD(O) != CD(O')**
  - Subject may not write to any other CD after reading from one

# Reading with Brewer-Nash

- **S may read O iff**
  **S has never read any O' such that**
  **COI(O) = COI(O') and CD(O) != CD(O')**
- If S has never read anything, S has free choice of what to read next
- Once S does read object from CD1 in COI1, a wall is erected around S
  - Cannot read other CDs from same COI
  - But can read from different COI
- If S does read from CD2 in COI2, wall changes shape
  - CD1 and CD2 inside wall
  - All other CDs from COI1 and COI2 outside the wall

# Writing with Brewer-Nash

- **S may write O iff**
  **S has never read any O' such that**
  CD(O) != CD(O')
- If S has never read anything, S has free choice of what to write
- If S has read from CD1, S may write only to CD1
- If S has read from CD2, S may not write at all
  - e.g. read from Bank of America and Exxon Mobil:
    - Now cannot write anywhere
    - Writing to Bank of America could leak info about Exxon Mobil and vv.
- Seems overly prohibitive…

# Users with Brewer-Nash

- A **subject** who has read two CDs may not write
- But that need not be true of a **user**
- Track read objects for:
  - user over its lifetime
  - subject over its lifetime (which is shorter than user)
  - distinguish what user has learned vs. what subject has learned
- As with MLS, user can choose to login at lower security level
  - **Attenuation of privilege:** give up the subject's right to read from CDs that have previously been read by user
  - Subject assigned that security level
  - So user could have multiple subjects with different security levels

# Users with Brewer-Nash

**Example:** Jane has read CD1 from COI1 and nothing from COI2

- Jane could login
  - with right to read CD1
  - or without that right
- Then subject on behalf of Jane reads CD2 from COI2: that is recorded for Jane as well and influences future subjects of hers
- Can Jane's subject write?
  - With right to read CD1: no
  - Without right: yes
- Jane's subject always prohibited from reading CD1' from COI1, regardless of whether right to read CD1 is enabled

So if user wants to work with different CDs, they can! Just disable access to the rest.

# RBAC

role-based access control

# Role-based Access Control

- enterprises and institutions are typically organised in roles
- different roles are granted different privileges
- roles:
  - student in a study program
  - teacher in a study program
  - teacher in another university
  - external examiner
- roles are more stable than users in a company
  - So they are a better candidate for authorization

RBAC

# “DAC for institutions”

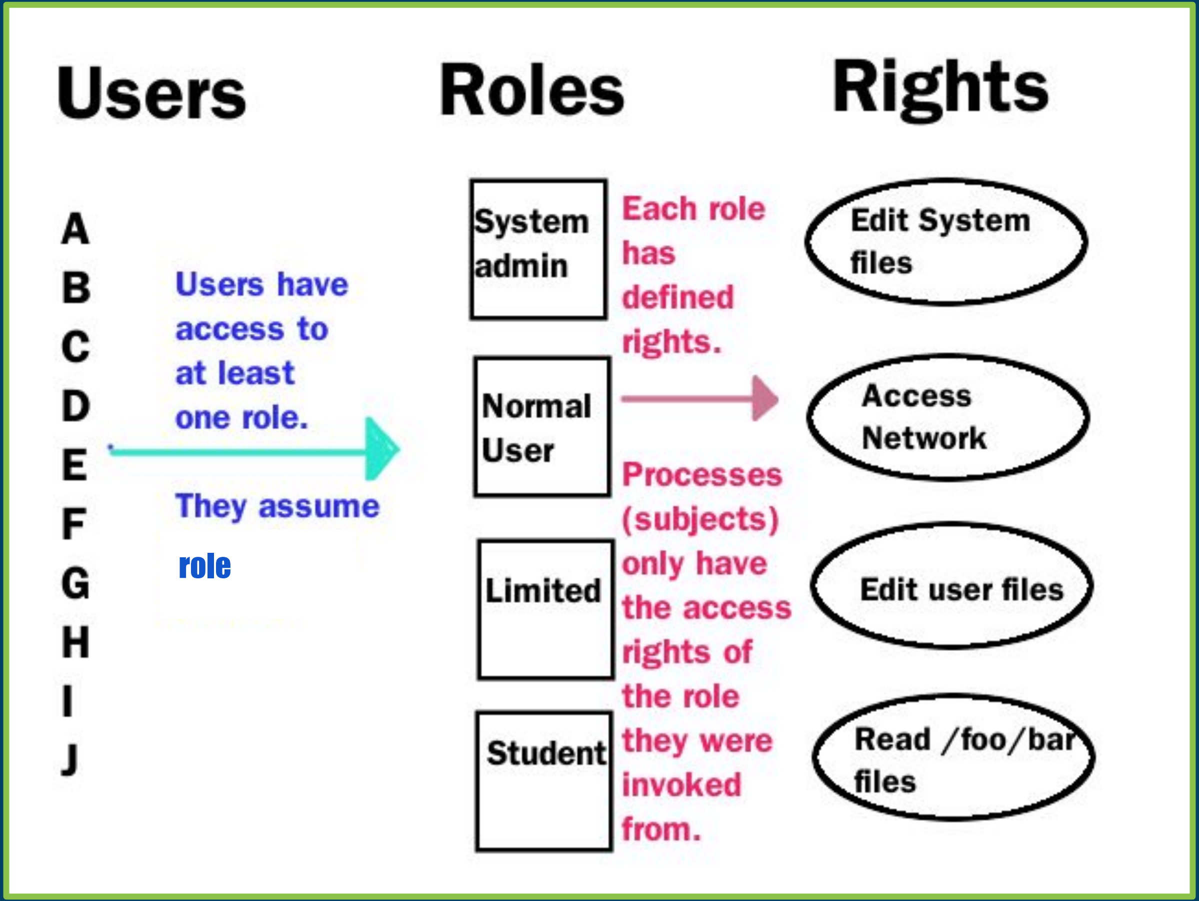
Why RBAC?

- goals of individuals may not be aligned
  to those of an institution
  (e.g., company, university, governmental, ...)

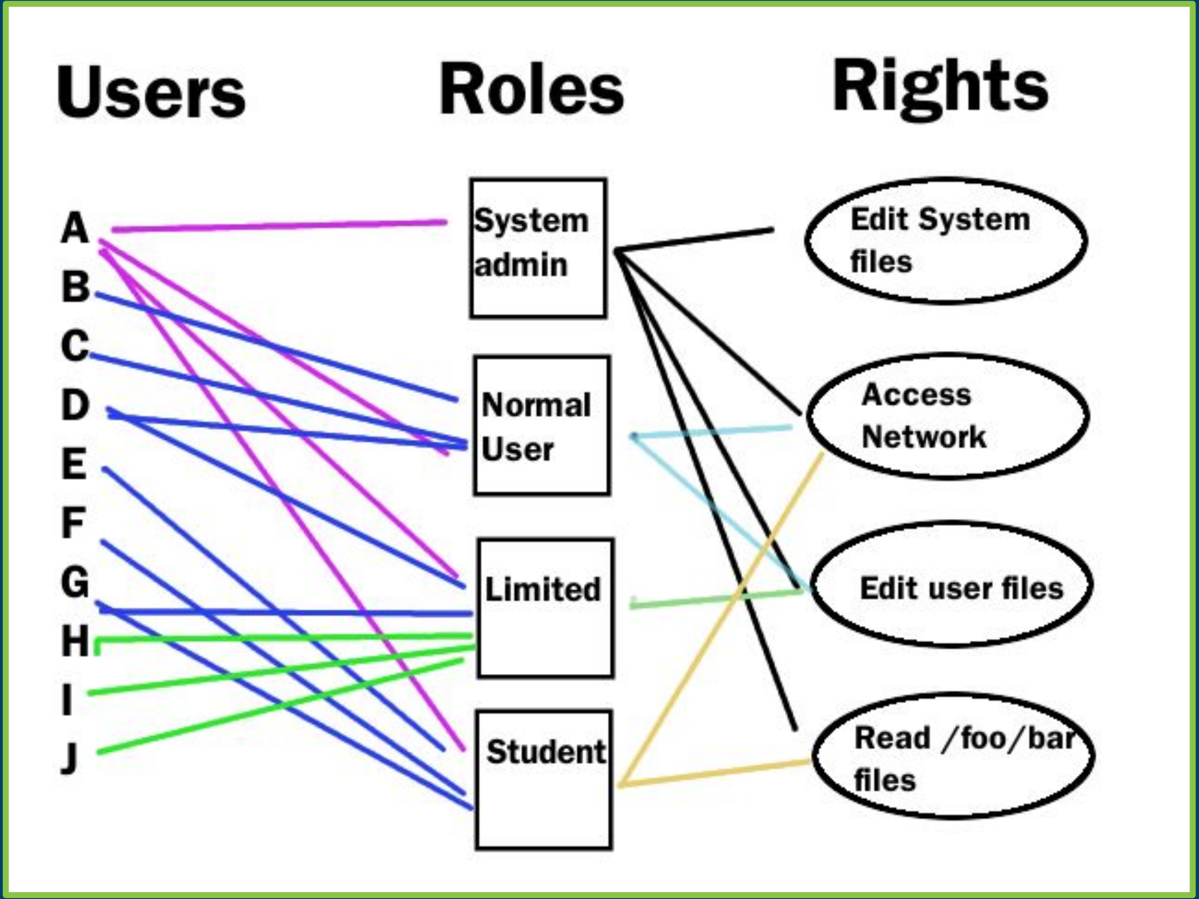
RBAC solution:

- set institutional rules that cannot be modified.

# RBAC


Users
Roles
Rights
A
B
C
D
E
F
G
H
I
J
Users have access to at least one role.
They assume
role
System admin
Normal User
Limited
Student
Each role has defined rights.
Processes (subjects) only have the access rights of the role they were invoked from.
Edit System files
Access Network
Edit user files
Read /foo/bar files

# RBAC


Users
Roles
Rights
A
B
C
D
E
F
G
H
I
J
System admin
Normal User
Limited
Student
Edit System files
Access Network
Edit user files
Read /foo/bar files

RBAC

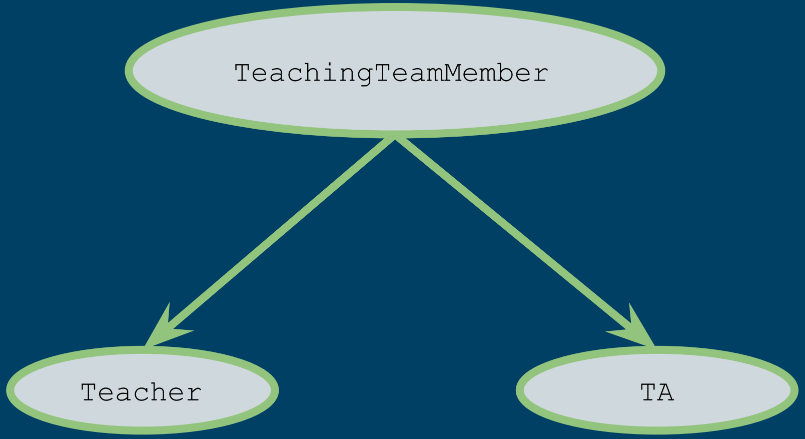
# Roles

- each user in the system is assigned a set of roles
  - UserRoles(C) = {`teacher, examiner`}
- each role is assigned a set of privileges
  - RolePrivs(`teacher`) = “read and write lecture notes”
  - RolePrivs(`student`) = “read lecture notes”
- users can have multiple roles active at a time;
  hence having all the privileges in each individual role
  - ActiveRole(C) = {`student, TA`}
  - RolePrivs(ActiveRole(C)) $= \cup_{R \in \text{ActiveRole}(C)}$ RolePrivs(R)

RBAC

# Roles Hierarchy

- roles form a hierarchy
  - The hierarchy is a partial order



- all privileges of the parents are transferred to the children

RolePrivs(`Teacher`) = RolePrivs(`TeachingTeamMember`) ∪ {...}

(teacher specific privileges)

# Role Constraints

- constraints may be added to role assignment in order to guarantee certain desired properties
- example
  - mutually exclusive roles (e.g., `studentAIS` and `teacherAIS`)

$$C1{:}\ \texttt{studentAIS} \notin \text{UserRoles}(\texttt{U}) \ \lor \ \texttt{teacherAIS} \notin \text{UserRoles}(\texttt{U})$$

- doable at any level of granularity (depends on the reference monitor)
  - limit active roles
  - constraints on the roles hierarchy
  - time (e.g., role R is disallowed from 8am to 10pm)
  - location (e.g., role R is only allowed when being at ITU)
  - ...

RBAC

# Roles Vs Groups

- role & group
  - set of users, that are assigned privileges

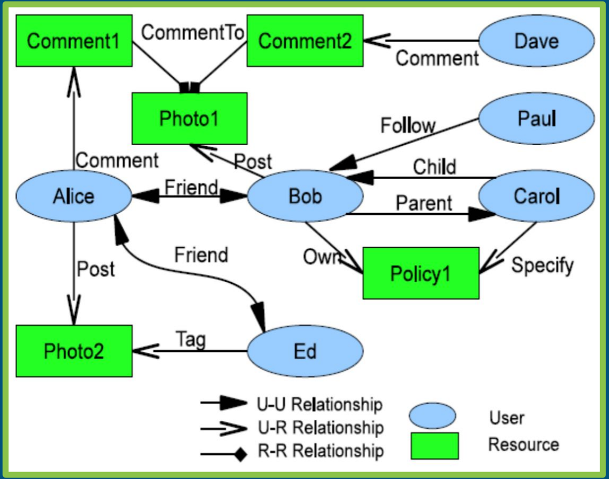
- differences?
  - roles may be active or inactive
  - roles form a hierarchy

- privileges handled resource-owner OR organization.
  role assignment handled by organization.
  - so, not DAC, but more...

# other

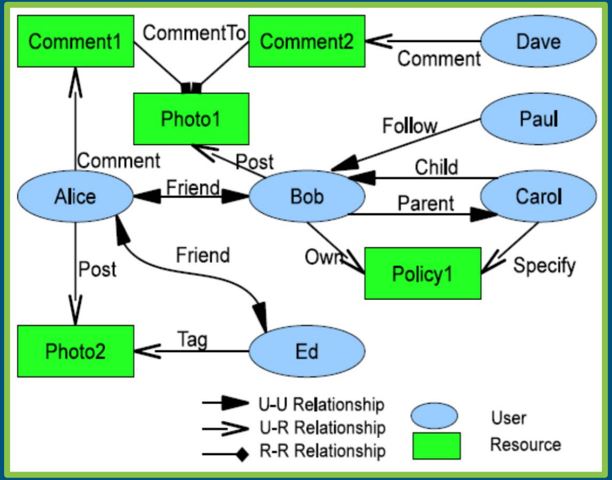
access control models

# Relationship-Based AC



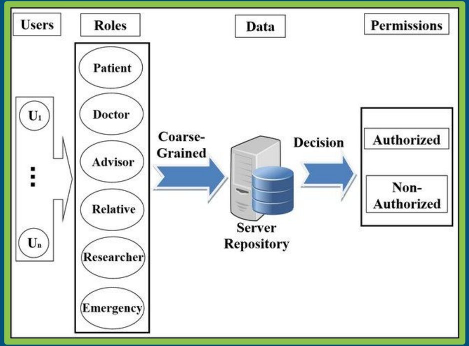
- online social networks
- people define audience of items based on social connections
  - my friends can access my posts
- DAC or MAC?

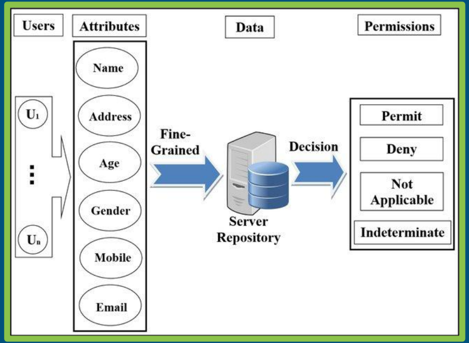
# Relationship-Based AC



- online social networks
- people define audience of items based on social connections
  - my friends can access my posts
- ReBAC not sufficient; we would need an role for each relationship (friends or Alice, friends of Bob)
  - relationships keep growing; requires a dynamic set of roles

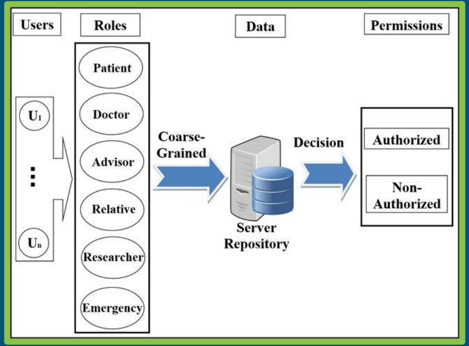
# Attribute-Based AC

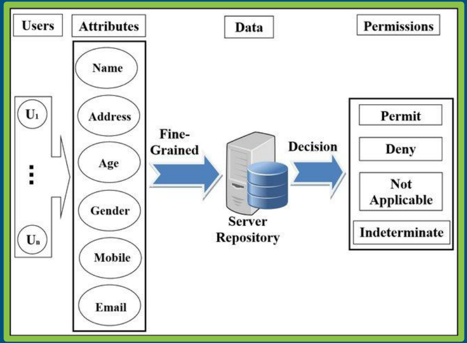




- each object has a set of attributes associated to it
  - defined by the organization
  - age, gender, *role*, creation time, ...
- access control policies depend on those attributes
  - defined by the organization or owner of resources
- directly compatible with attribute-based encryption (ABE): user’s secret key depends on his/her attributes
- DAC or MAC?

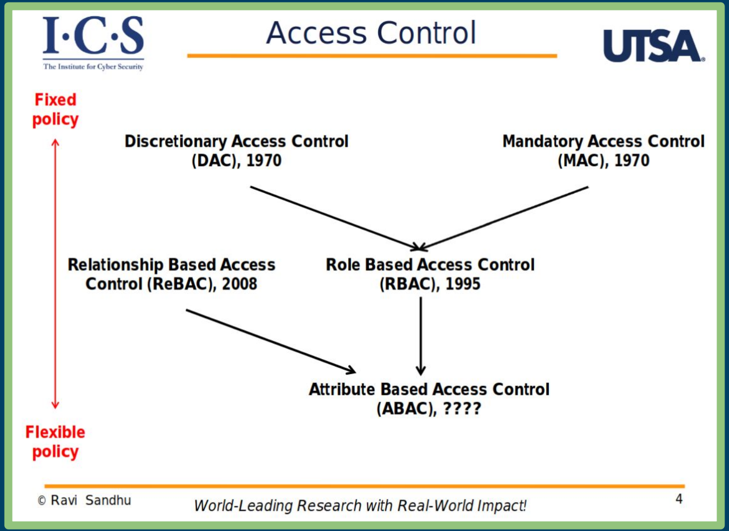
# Attribute-Based AC





- each object has a set of attributes associated to it
  - defined by the organization
  - age, gender, *role*, creation time, ...
- access control policies depend on those attributes
  - defined by the organization or owner of resources
- directly compatible with attribute-based encryption (ABE): user's secret key depends on his/her attributes
- ReBAC not sufficient; we would need a role for each attribute condition in a policy
  - also dynamic, which implies a dynamic set of roles

# Expressiveness of AC models (ReBAC, ABAC)



I·C·S The Institute for Cyber Security

Access Control

UTSA

Fixed policy

Discretionary Access Control (DAC), 1970

Mandatory Access Control (MAC), 1970

Relationship Based Access Control (ReBAC), 2008

Role Based Access Control (RBAC), 1995

Attribute Based Access Control (ABAC), ????

Flexible policy

© Ravi Sandhu

World-Leading Research with Real-World Impact!

4

Source: https://www.profsandhu.com/cs5323_s17/L6.pdf

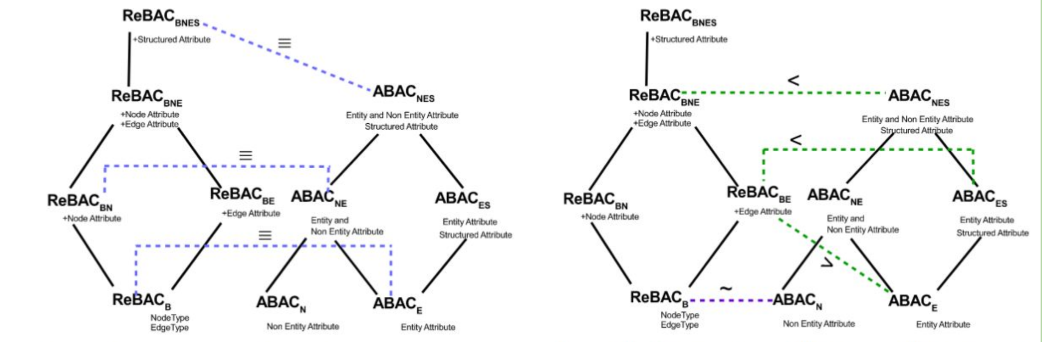
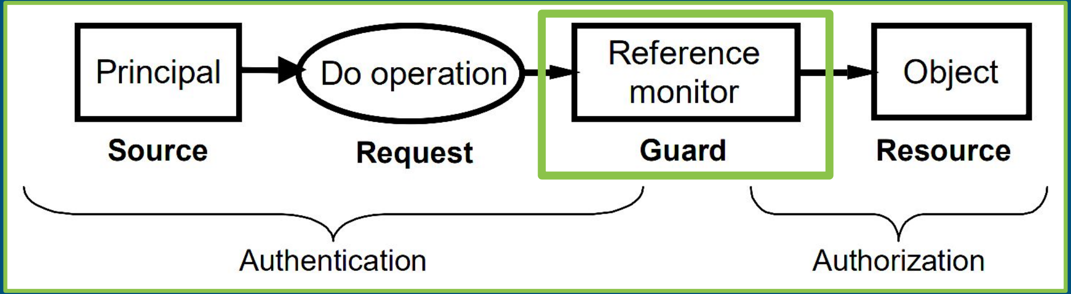


Figure 11: *Equivalence of ReBAC and ABAC Structural Classification*

Figure 12: *Non-Equivalence of ReBAC and ABAC Structural Classification*

Source: https://profsandhu.com/confrnc/misconf/codaspy17-tahmina.pdf

# Reference Monitor



Reference Monitor - Recap (“Assurance”)

# Security Policy

**security** a system is secure iff it

- does what it *should*,
- and nothing more.

**policy** stipulates what should and should not be *done*.

**format** “The system shall
{ prevent, detect } [ *action* ]
{ on, to, with } [ *asset* ].”

**enforcing** AC policies:
<u>complete mediation</u> mechanism
+ AC matrix. let’s see mechanism.

# Reference Monitor

**reference monitor:** piece of SW that checks each reference made by subjects to objects.

(note: not the only way to get complete mediation. e.g. program analysis)

**TCB:** trusted computing base

all SW/HW components that must function correctly for the system to implement its security policy.

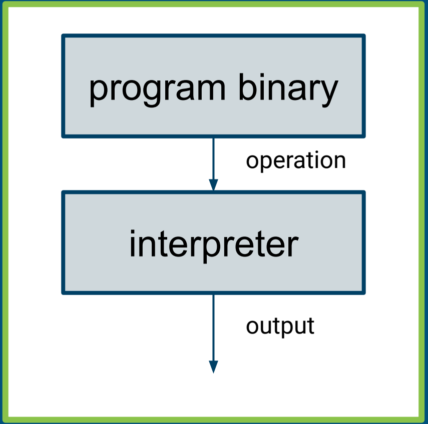
TCB compromised ⇒ system compromised. keep the TCB small.
Reference monitor in TCB. a good monitor is small.

OS kernels *can* be small (microkernel). In practise, large (thus large TCB).

Reference Monitor

approach 1:

# Interpreter



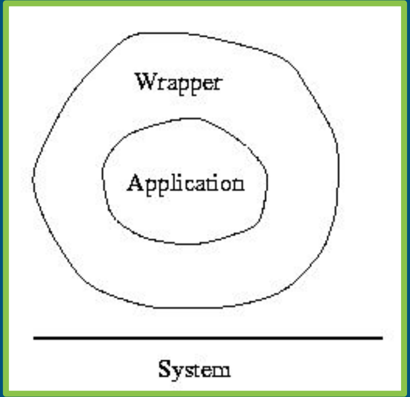
program interpreted by monitor.

each instruction only executed if monitor OKs it (conforms to policy)

- broad
  e.g. do not execute two MOVs in a row, do not write to that section of disk, …
- slow
  1 program instruction ⇒
  12 monitor instructions?

approach 2:

# Wrapper



intercepts (& interprets / redirects) only *some* program instructions.

- potentially faster

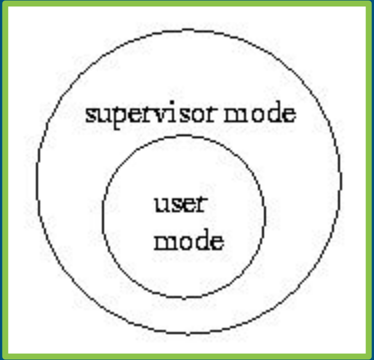
  monitor overhead only for caught instructions.
- wrapper can only restrict ops that it sees.

  cannot enforce all policies.

Reference Monitor

approach 3:

# Hardware



**recall:**

- each process has its memory.
- exists state not associated w/ any process (e.g. I/O registers).

instructions manipulating that state distinct from other instructions.

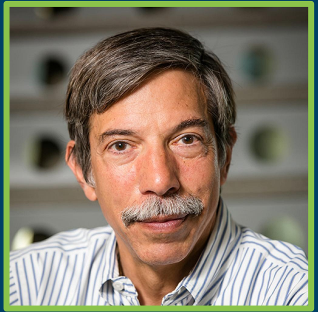
- user mode process state
- supervisor mode any state

restrict processes to only execute certain instruction sequences in supervisor mode.

⇒ OS is a wrapper!

Reference Monitor

# Enforceable Security Policies



$$\forall \overset{\text{state}}{s} \in S \;.\; \overset{\text{property}}{p}(s)$$

monitor can only accept/reject *current trace*; knows nothing about other possible traces (safety properties).

```
x := false; if (secret) { x := true }; out public x
```

‘secret == false’ ⇒ monitor does not reject the output at the end.

monitors are **not** the be-all end-all of enforcements! They are limited!

(fortunately, we also have other approaches; stay tuned)

Reference Monitor

# Secure by Transformation

**SW fault isolation**

program, transformed to satisfy policy.

examples follow (not just monitors):

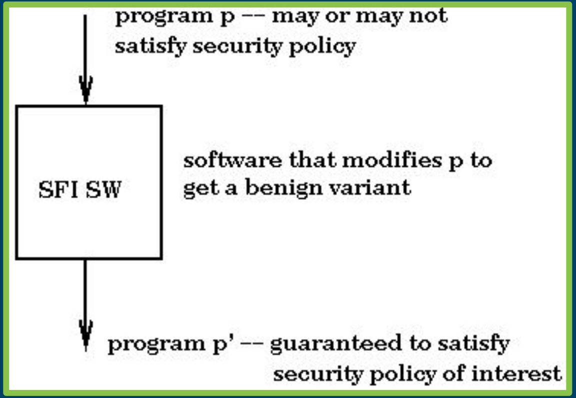
**inlined reference monitor**

program, transformed to include a monitor.

old program misbehaves ⇒ new program self-destructs.

**secure multi-execution**

replicate program, one run per principal.

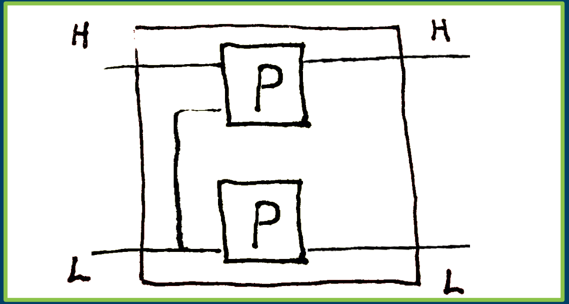


program p –– may or may not satisfy security policy

SFI SW

software that modifies p to get a benign variant

program p' –– guaranteed to satisfy security policy of interest



# summary

# Summary

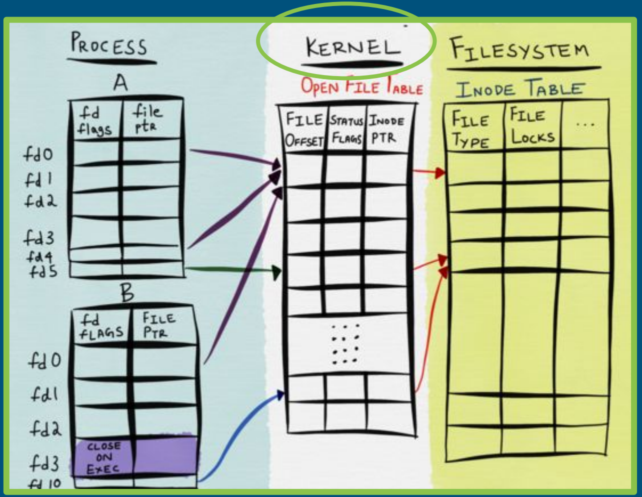
from authentication to authorization:
access control! (AC) models:

| | |
|---|---|
| ● DAC | discretionary AC |
| ○ ACL | AC lists |
| ○ Capabilities | discretionary AC |
| ● MAC | mandatory access control |
| ○ MLS | multi-level security [Bell-Lapadua, Biba] |
| ○ commerce | [Brewer-Nash] |
| ● RBAC | role-based AC |
| ● other models | ABAC, ReBAC |

enforcement: reference monitor

Summary

# Ambient Authority



in your OS,

- a process does not (need not, cannot) name an authority that justifies ops on the world around it.

OS kernel

this **ambient authority** is implied, and trusted.

on the Internet (API, services talk to services), we have **no such authority**.

- Facebook: “Hi Twitter. So, you want to post on Bob’s wall? **By who’s authority?**”

solution: **capabilities**. include the authority in the request, so system can check & authorize.

how: **bearer tokens**.