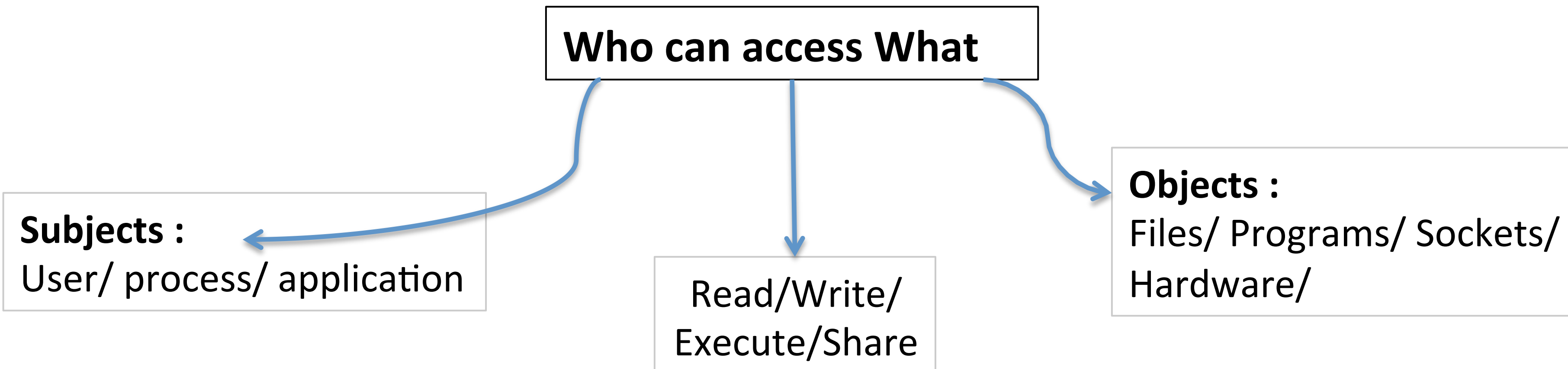
Access Control

Chester Rebeiro

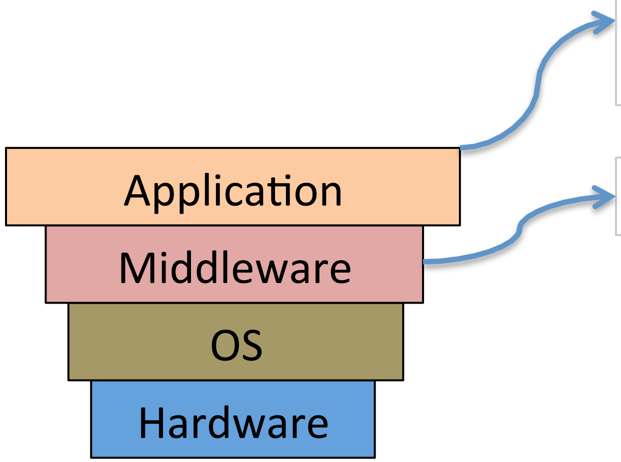
Indian Institute of Technology Madras

Access Control

(the tao of achieving confidentiality and integrity)



Access Control
(number of levels)



Elaborate and complex. Many people may be involved
Multiple roles. Hundreds of transactions feasible

Eg. DBMS. Who gets to access what fields in the DB

Moving from Hardware to Application

- More aspects to control
  - More subjects and objects involved
  - Inter-relationship becomes increasingly difficult
- Complexity increases
- Reliability Decreases
  - More prone to loopholes that can be exploited

Hardware Access Control

- **Policies**
  - Must protect OS from applications
  - Must protect applications from others
  - Must prevent one application hogging the system

  (first two ensure confidentiality and integrity, the third ensures availability)
- **Mechanisms**
  - Paging unit
  - Privilege rings
  - Interrupts

Access Control at OS Level

Policies

- Only authenticated users should be able to use the system
- One user's files should be protected from other users
  (not present in older versions of Windows)
- A Process should be protected from others
- Fair allocation of resources (CPU, disk, RAM, network) without starvation

Mechanisms

- User authentication
- **Access Control Mechanisms for Files (and other objects)**
- For process protection leverage hardware features (paging etc.)
- Scheduling, deadlock detection / prevention to prevent starvation

Access Control for Objects in the OS

- Discretionary (DAC)
  - Access based on
    - Identity of requestor
    - Access rules state what requestors are (or are not) allowed to do
  - Privileges granted or revoked by an administrator
  - Users can pass on their privileges to other users
  - The earliest form called Access Matrix Model

Access Matrix Model

- By Butler Lampson, 1971 (Earliest Form)
- Subjects : active elements requesting information
- Objects : passive elements storing information
  - Subjects can also be objects

objects

subjects

rights

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Other actions : ownership (property of objects by a subject),
control (father-children relationships between processes)

Butler Lampson, “Protection”, 1971

A Formal Representation of Access Matrix

- Define an access matrix : $A[X_{s_i}, X_{o_j}]$
- Protection System consists of
  - **Generic rights :** $R = \{r_1, r_2, \cdots, r_k\}$ **thus** $A[X_{s_i}, X_{o_j}] \subseteq R$
  - **Primitive Operations** $O = \{op_1, op_2, \cdots, op_n\}$

enter r into $A[X_{s_i}, X_{o_j}]$
delete r from $A[X_{s_i}, X_{o_j}]$
create subject X_s
create object X_o
destroy subject X_s
destroy object X_o

objects

subjects

generic rights

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Michael A. Harrison, Walter L. Ruzzo, Jeffrey D. Ullman, Protection in Operating Systems, 1974

A formal representation of Access Matrix Model

- Commands : conditional changes to ACM

command $\alpha(X_1, X_2, \cdots, X_n)$
  if r_1 in $A[X_{s_1}, X_{o_1}]$ and
    r_2 in $A[X_{s_2}, X_{o_2}]$ and
    r_3 in $A[X_{s_3}, X_{o_3}]$ and
    $\vdots \quad \vdots \quad \vdots$
    r_3 in $A[X_{s_3}, X_{o_3}]$
  then op_1
    op_2
    op_3
    $\vdots \ \vdots$
  end

access matrix $A[X_{s_i}, X_{o_j}]$

Generic rights $R = \{r_1, r_2, \cdots, r_k\}$

Primitive operations $O = \{op_1, op_2, \cdots, op_n\}$

enter r into $A[X_{s_i}, X_{o_j}]$
delete r from $A[X_{s_i}, X_{o_j}]$
create subject X_s
create object X_o
destroy subject X_s
destroy object X_o

Example Commands

$$
\begin{array}{l}
\text{command } \alpha(X_1, X_2, \cdots, X_n) \\
\quad \text{if } r_1 \text{ in } A[X_{s_1}, X_{o_1}] \text{ and} \\
\qquad r_2 \text{ in } A[X_{s_2}, X_{o_2}] \text{ and} \\
\qquad r_3 \text{ in } A[X_{s_3}, X_{o_3}] \text{ and} \\
\qquad \vdots \qquad \vdots \qquad \vdots \\
\qquad r_3 \text{ in } A[X_{s_3}, X_{o_3}] \\
\quad \text{then} \qquad op_1 \\
\qquad op_2 \\
\qquad op_3 \\
\qquad \vdots \; \vdots \\
\quad \text{end}
\end{array}
$$

```
command CREATE(process, file)
    create object file
    enter own into (process, file)
end
```

Create an object

```
command CONFER_r (owner, friend, file)
    if own in (owner, file)
    then enter r into (friend, file)
end
```

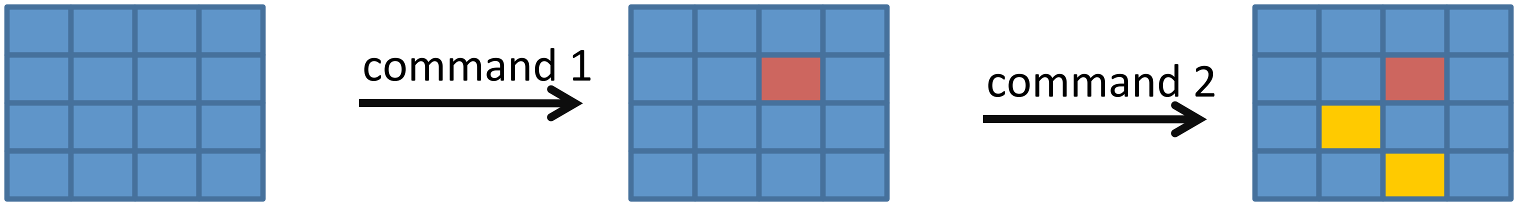
Confer 'r' right to a friend for the object

```
command REMOVE_r (owner, exfriend, file)
    if own in (owner, file) and
        r in (exfriend, file)¹
    then delete r from (exfriend, file)
end
```

Owner can revoke Right from an 'ex'friend

States of Access Matrix

- A protection system is a **state transition system**



Safety

- Suppose a subject **s** plans to give subjects **s'** right **r** to object **o**.
  - with **r** entered into **A[s',o]**, is such that **r** could subsequently be entered somewhere new.
  - If this is possible, then the system is unsafe

Unsafe State (Example)

- Consider a protection system with two commands

command $CONFER_{execute}(S, S', O)$
  if o in $A[S, O]$ then
    enter x in $A[S', 0]$
  end

command $MODIFY\_RIGHT(S, O)$
  if x in $A[S, O]$ then
    enter w in $A[S, O]$
  end

- Scenario: Bob creates an application (object). He wants it to be executed by all others but not modified by them
- The system is unsafe due to the presence of MODIFY_RIGHT in the protection system.
  - Alice could invoke MODIFY_RIGHT to get modification rights for the application

Safety Theorem

- Given an initial state of the matrix (say $\mathbf{A_0}$) and a right '**r**', we say that the state $\mathbf{A_0}$ is unsafe if there exists a state $\mathbf{A_i}$ such that,

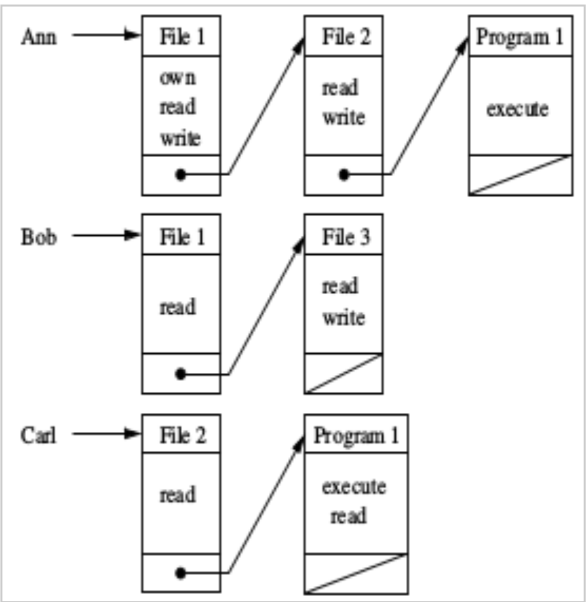
  1. $\mathbf{A_i}$ is reachable from $\mathbf{A_0}$
     - There are a sequence of transitions (commands) that would take the state from $\mathbf{A_0}$ to $\mathbf{A_i}$

  2. $\mathbf{A_i}$ leaks 'r'

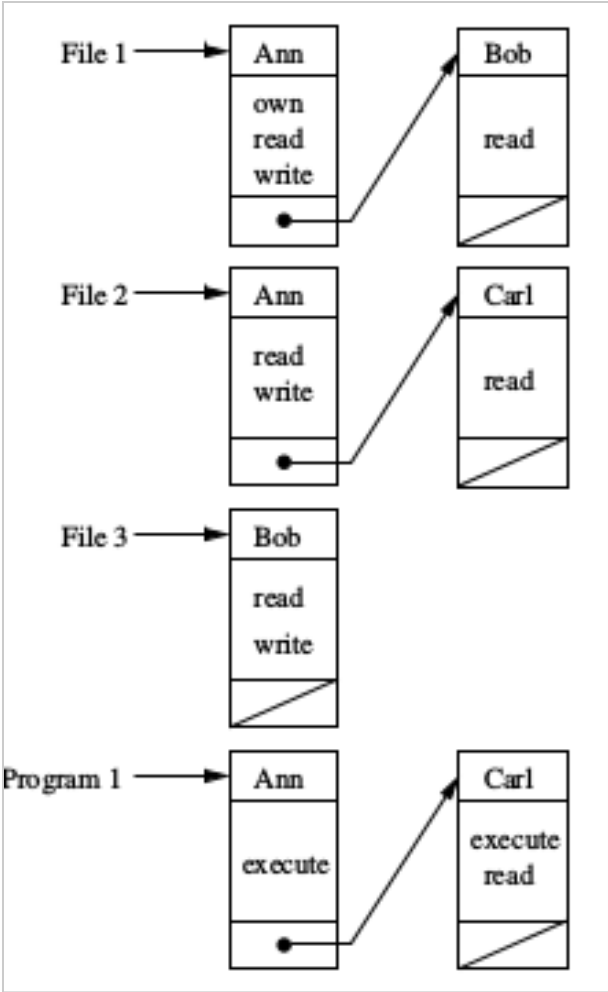
Determining if a system is safe is undecidable

Implementation Aspects

Capabilities



Access Control List



Capabilities : ticket
ACL : My name is in the list

Railway Reservation

Capability vs ACL

- Delegation

  CAP: easily achieved

  For example “Ann” can create a certificate stating that she delegates to “ Ted” all her activities from 4:00PM to 10:00PM

  ACL: The owner of the file should add permissions to ensure deligation

- Revocation

  ACL: Easily done, parse list for file, remove user / group from list

  CAP: Get capability back from process

  If one capability is used for multiple files, then revoke all or nothing

Unix Security Mechanism

- **Subject:** process
- **Objects:** files, directories, sockets, process, process memory, file descriptors

- Each process is associated with a user ID (32 bit integer) and group ID (32 bit user integer)
- The privileges of a process depends on the user ID and group ID

File Operations in Unix

Operations for a file

- Create
- Read
- Write
- Execute (does this imply read?)
- Ownership
- Change permissions

Operations for a directory

- Create
- Unlink / link
- Rename a file
- lookup

Permissions for files and directories

In inode :
uid, gid

	R	W	X
Owner	1	1	0
Group	1	0	0
Other	1	0	0

Change permissions by owner (same uid as the file)

For directories almost similar: linking / unlinking write permissions

X permission on a directory implies look up. You can look up a name but not read the contents of the directory

Additionally bits are present to specify type of file (like directory, symbolic link, etc.)

User IDs

- UID = 0 is root permissions
- setuid(user ID) →set the user id of a process. Can be executed only by processes with UID = 0
- setgid(group iD) → set the group id of a process
- Login process
  - At the time of login, the login process runs with uid=0
  - If user name and password is verified,
    - Use uid stored in /etc/passwd file to invoke setuid()
    - Invoke shell with the user's process ID
- setuid bit in inode
  - Allows a program to execute with the privileges of the owner of the file.

sudo / su

- used to elevate privileges
  - If permitted, switches uid of a process to 0 temporarily
  - Remove variables that control dynamic linking
  - Ensure that timestamp directories (/var/lib/sudo) are only writeable by root

```
chester@optiplex:~$ id
uid=1000(chester) gid=1000(chester) groups=1000(chester),4(adm),24(cdrom),27(sud
o),30(dip),46(plugdev),108(lpadmin),124(sambashare)
chester@optiplex:~$
chester@optiplex:~$ sudo id
[sudo] password for chester:
uid=0(root) gid=0(root) groups=0(root)
```

File Descriptors

- Represents an open file
- Two ways of obtaining a file descriptor
  - Open a file
  - Get it from another process
    - for example a parent process
    - Through shared memory or sockets
- If you have a file descriptor, no more explicit checks

Processes

- Operations
  - Create
  - kill
  - Debug (ptrace system call that allows one process to observe the control the other)
- Permissions
  - Child process gets the same uid and gid as the parent
  - ptrace can debug other processes with the same uid

Network Permissions in Unix

- Operations
  - Connect
  - Listening
  - Send/Receive data
- Permissions
  - Not related to UIDs. Any one can connect to a machine
  - Any process can listen to ports > 1024
  - If you have a descriptor for a socket, then you can send/receive data without further permissions

Problems with the Unix Access Control

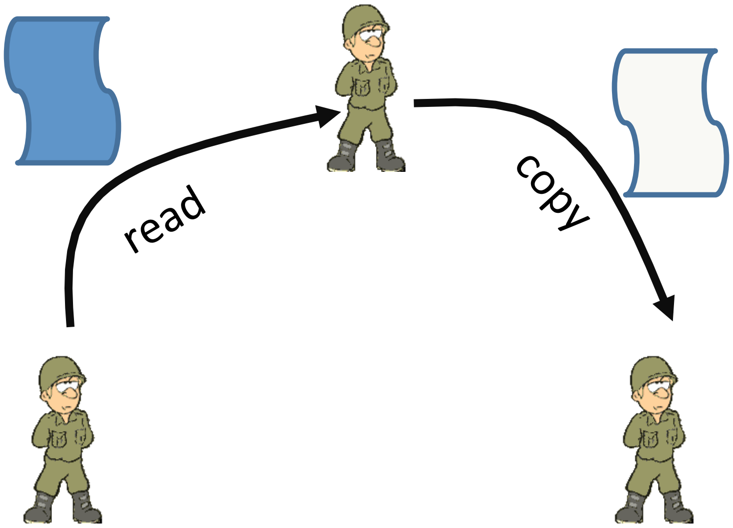
- **Root can do anything (has complete access)**
  - Can delete / modify files
    (FreeBSD, OSX, prevent this by having flags called append-only, undeletable, system → preventing even the root to delete)
  - Problem comes when (a) the system administrator is untrustable (b) if root login is compromised
- **Permissions based on uid are coarse-grained**
  - a user cannot easily defend himself against allegations
  - Cannot obtain more intricate access control such as
    "X user can run program Y to write to file Z"
  - Only one user and one group can be specified for a file.

Vulnerabilities in Discretionary Policies

- Discretionary policies only authenticate a user
- Once authenticated, the user can do anything
- Subjected to Trojan Horse attacks
  - A Trojan horse can inherit all the user's privileges
  - Why?
    - A trojan horse process started by a user sends requests to OS on the user's behalf

Drawback of Discretionary Policies

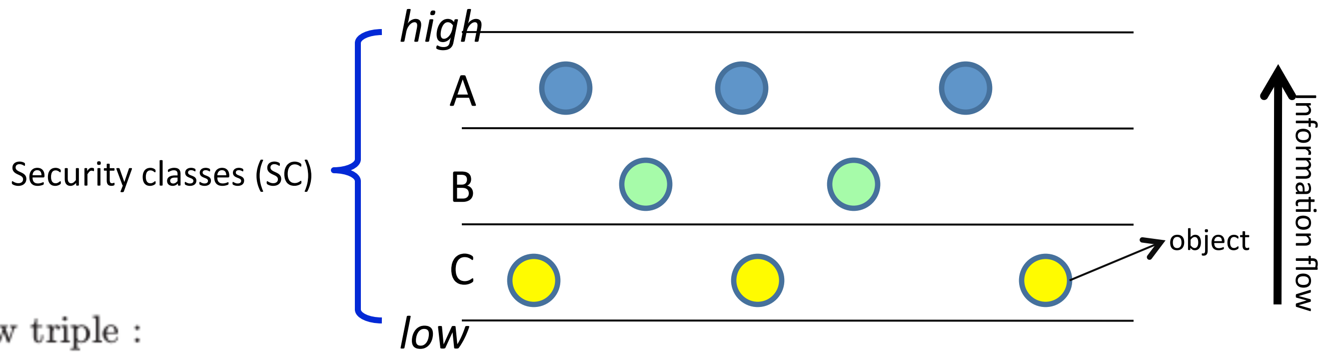
- It is not concerned with information flow
  - Anyone with access can propagate information



- Information flow policies
  - Restrict how information flows between subjects and objects

Information Flow Policies

- Every object in the system assigned to a security class (SC)



– Information flow triple :

$\langle SC, \rightarrow, \oplus \rangle$

$\rightarrow$ is the can flow relation

- $B \rightarrow A$: Information from B can flow to A
- $C \rightarrow B \rightarrow A$: Information flow
- $C \leq B \leq A$: Dominance relation

$\oplus$ is the join relation

- defines how to label information obtained by combining information from two classes
- $\oplus : SC \times SC \rightarrow SC$.

SC, $\rightarrow$, and $\oplus$ are fixed and do not change with time.

The SC of an object may vary with time

Ravi Sandhu, *Lattice Based Access Control Models*, 1993

Examples

- Trivial case (also the most secure)
  - No information flow between classes

$-\ SC = \{A_1\,(low), A_2, \cdots, A_n\,(high)\}$
$-\ A_i \rightarrow A_i \ (\text{for } i = 1 \cdots n)$
$-\ A_i \oplus A_i = A_i$

- Low to High flows only

$-\ SC = \{A_1\,(low), A_2, \cdots, A_n\,(high)\}$
$-\ A_j \rightarrow A_i \text{ only if } j \leq i \ (\text{for } i, j = 1 \cdots n)$
$-\ A_i \oplus A_j = A_i$

Exercises

- A company has the following security policy
  - A document made by a manager can be read by other managers but no workers
  - A document made by a worker can be read by other workers but no managers
  - Public documents can be read by both Managers and Workers

- What are the security classes?
- What is the flow operator?
- What is the join operator?

Exercises

- A company has the following security policy
  - A document made by a manager can be read by other managers but no workers
  - A document made by a worker can be read by other workers but no managers
  - Public documents can be read by both Managers and Workers

$-\ SC = \{P\,(low), W, M\,(high)\}$

$-\ P \rightarrow M,\ P \rightarrow W,\ W \rightarrow W,\ M \rightarrow M$

$-\ P \oplus M \rightarrow M,\ P \oplus W \rightarrow W,\ M \oplus M \rightarrow M,\ W \oplus W \rightarrow W$

Mandatory Access Control

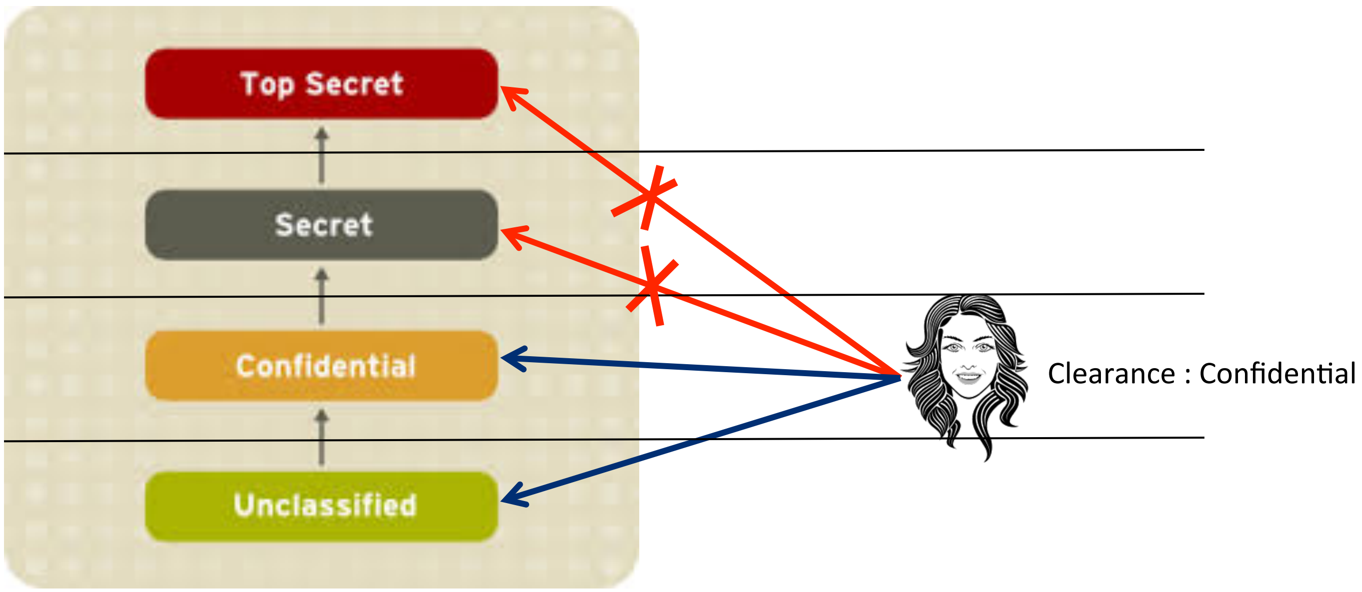
- Most common form is multilevel security (MLS) policy
  - Access Class
    - Objects need a **classification level**
    - Subjects needed a **clearance level**
  - A subject with *X* clearance can access all objects in X and below X but not vice-versa
  - Information only flows upwards and cannot flow downwards

Bell-LaPadula Model

- Developed in 1974
- Objective : Ensure that information does not flow to those not cleared for that level
- Formal model for access control
  - allows formally prove security
- Four access modes:
  - read, write, append, execute
- Three properties (MAC rules)
  - No read up (simple security property (ss-property))
  - No write down (*-property)
  - ds property : discretionary security property (every access must be allowed by the access matrix)

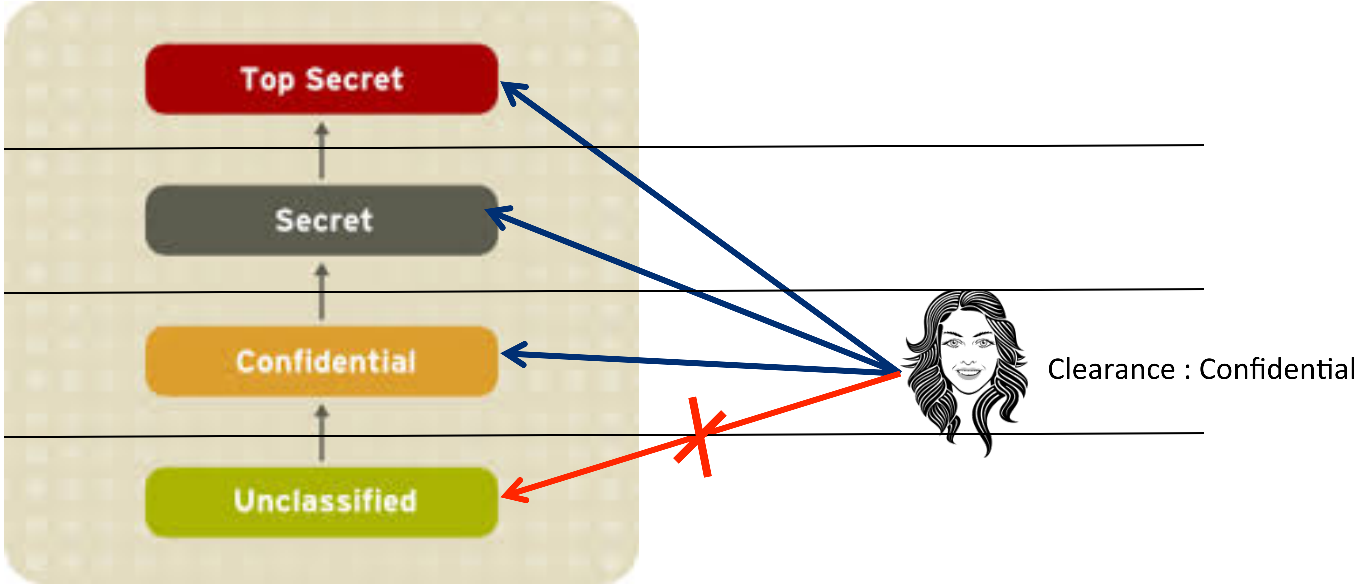
D. E. Bell and L. J. LaPadula, *Secure Computer System: Unified*

No read up



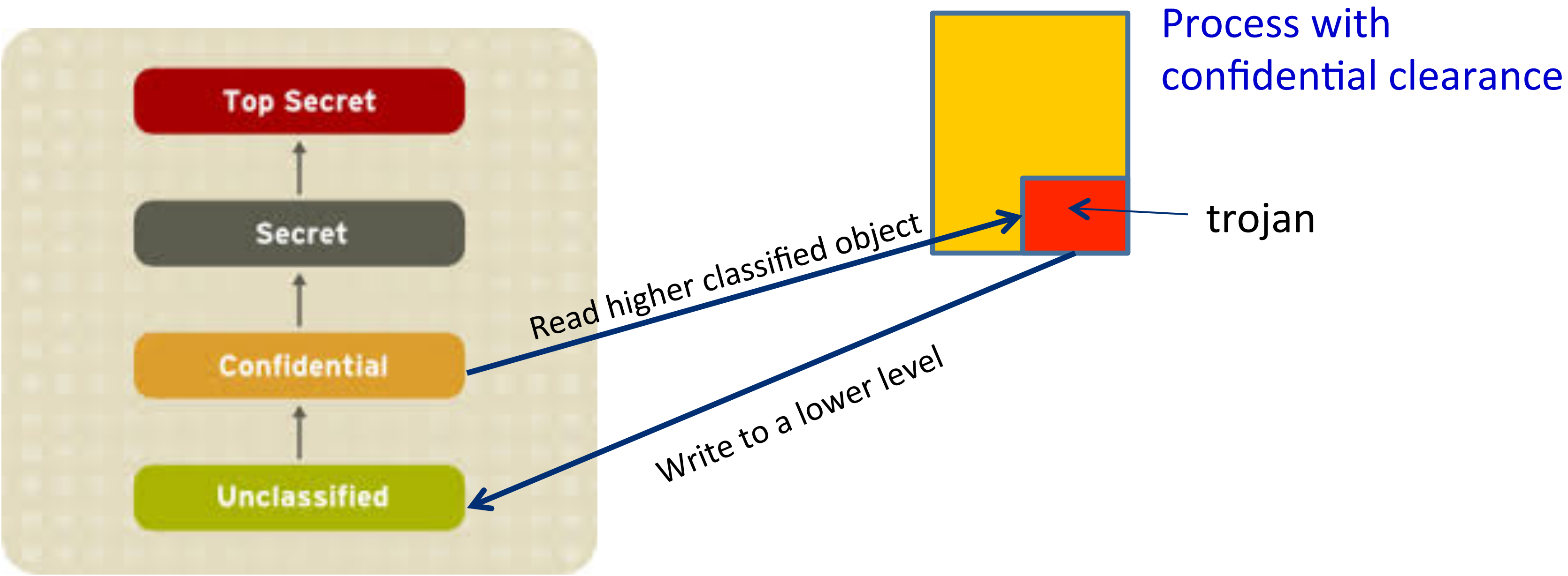
- Can only read confidential and unclassified files

No Write Down



- Cannot write into an unclassfied object

Why No Write Down?



- A process inflected with a trojan, could read confidential data and write it down to unclassified
- We trust users but not subjects (like programs and processes)

ds-property

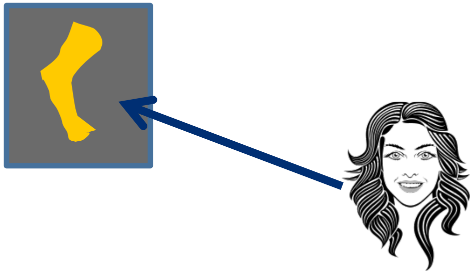
- Discretionary Access Control
  - An individual may grant access to a document he/she owns to another individual.
  - However the MAC rules must be met

MAC rules over rides any discretionary access control. A user cannot give away data to unauthorized persons.

Limitations of BLP

- Write up is possible with BLP
- Does not address Integrity Issues

file with classification *secret*



Clearance : Confidential

User with clearance can modify a secret document
BLP only deals with confidentiality. Does not take care of integrity.

Limitation of BLP (changing levels)

- Suppose someone changes an object labeled ***top secret*** to ***unclassified***.
  - breach of confidentiality
  - Will BLP detect this breach?
- Suppose someone moves from clearance level top secret to unclassified
  - Will BLP detect this breach?

Need an additional rule about changing levels

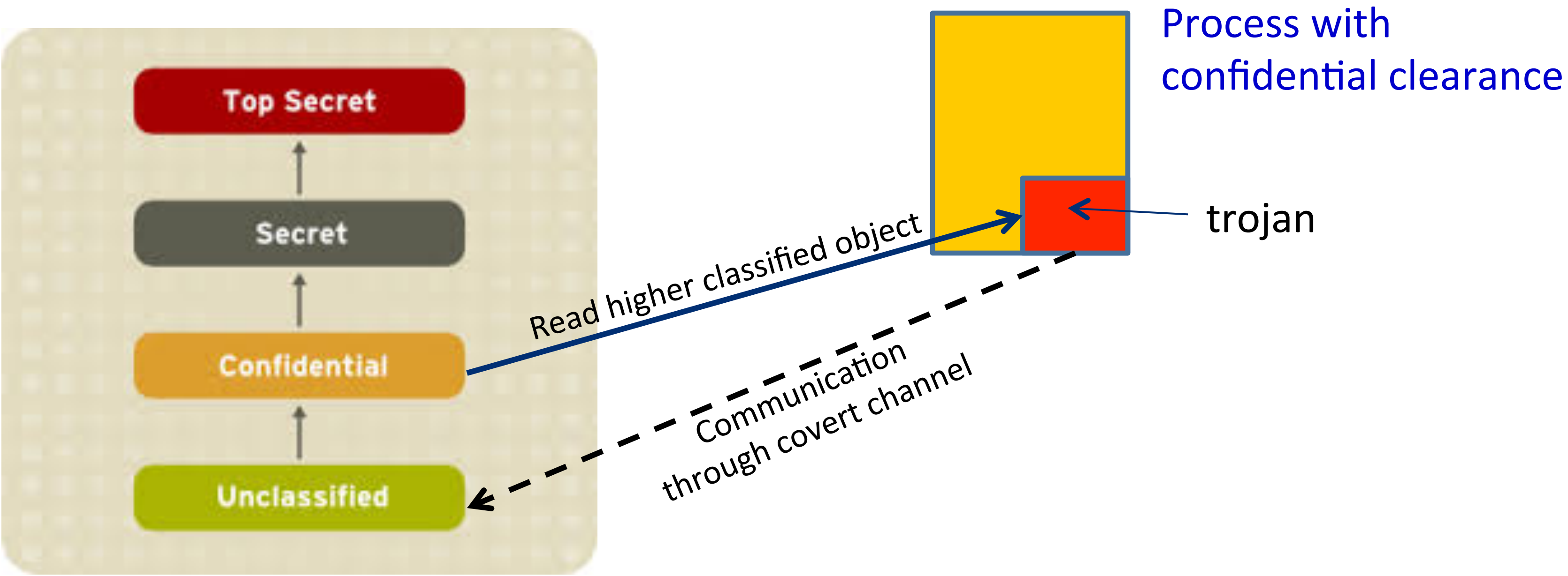
Tranquility

- **Strong Tranquility Property:**
  - Subjects and objects do not change label during lifetime of the system
- **Weak Tranquility Property:**
  - Subjects and objects do not change label in a way that violates the *spirit* of the security policy.
  - Should define
    - How can subjects change clearance level?
    - How can objects change levels?

Principle of Least Privilege

- Every subject has access to the minimum amount of information and resources that are necessary
- Useful for implementing weak tranquility.

Limitations of BLP (Covert Channels)



- Covert channels through system resources that normally not intended for communication.
- covert channel examples:
  page faults, file lock, cache memory, branch predictors , rate of computing, sockets
- Highly noisy, but can use coding theory to encode / decode information through noisy channels

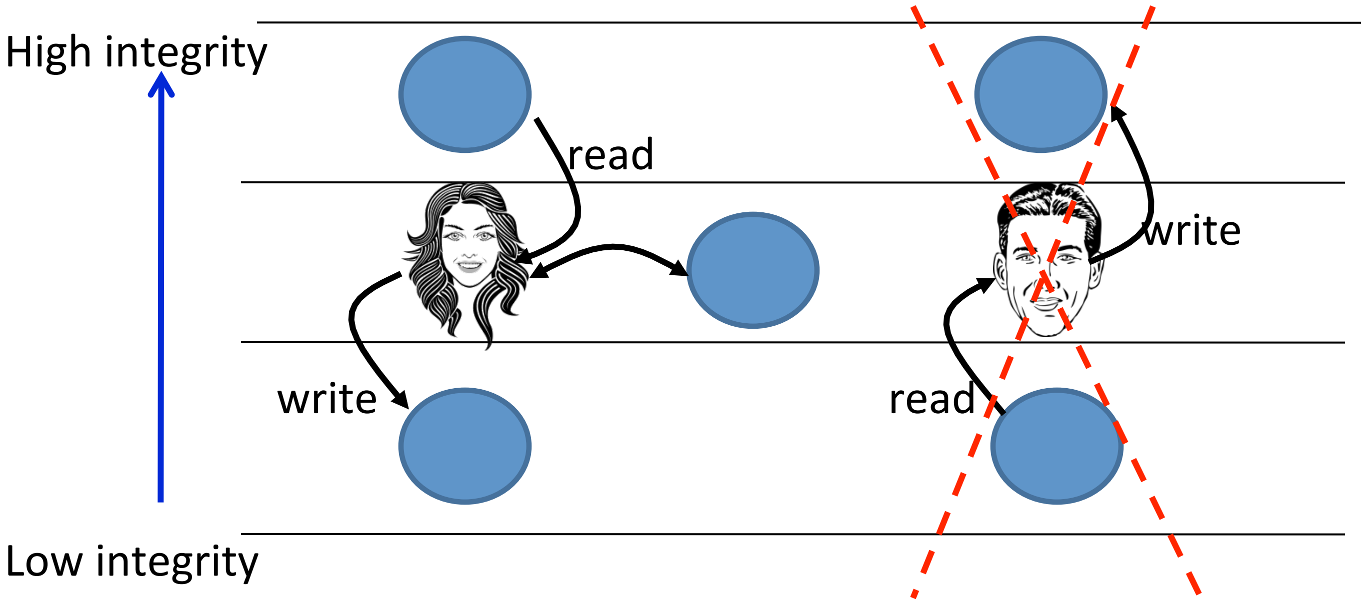
Covert Channels

- Identifying: Not easy because simple things like the existence of a file, time, etc. could be a source for a covert channel.
- Quantification: communication rate (bps)
- Elimination: Careful design, separation, characteristics of operation (eg. rate of opening / closing a file)

Biba Model

- Bell-LaPadula upside down
- **Ignores confidentiality and only deals with integrity**
- Goals of integrity
  - Prevent unauthorized users from making modifications to an object
  - Prevent authorized users from making improper modifications to an object
  - Maintain consistency (data reflects the real world)
- Incorporated in FreeBSD

BIBA Properties (read up / write down)



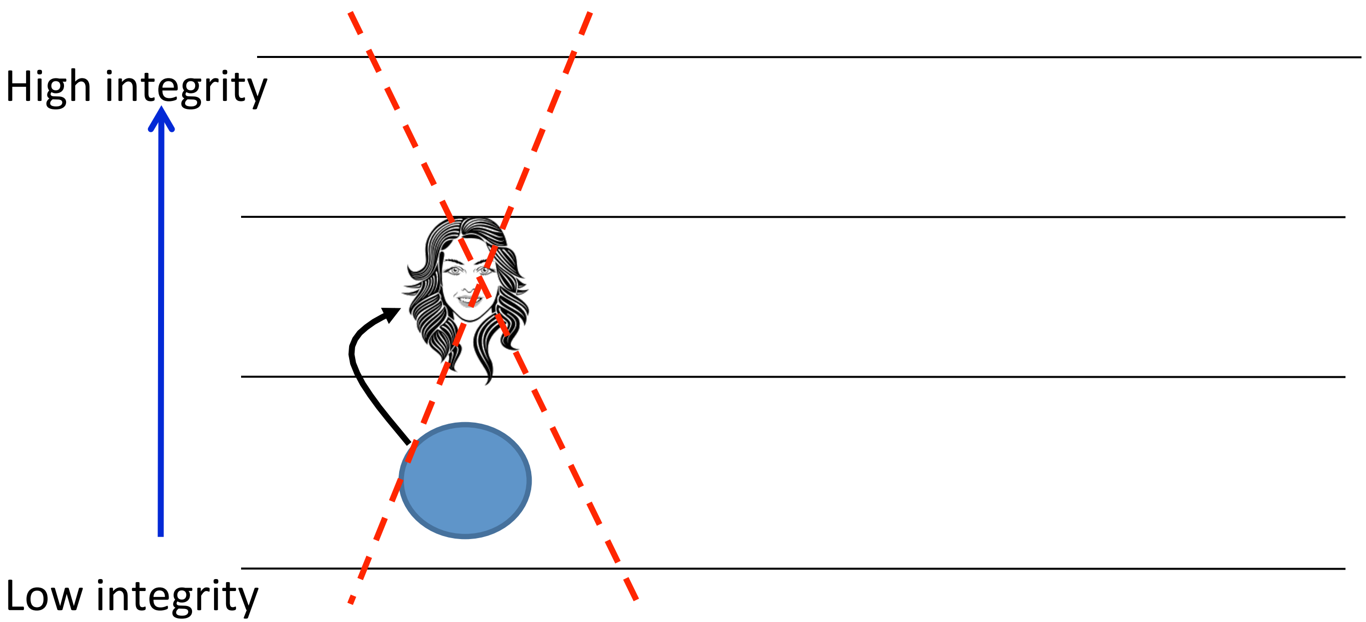
Properties

No read down : Simple Integrity Theorem

No write up : * Integrity Theorem

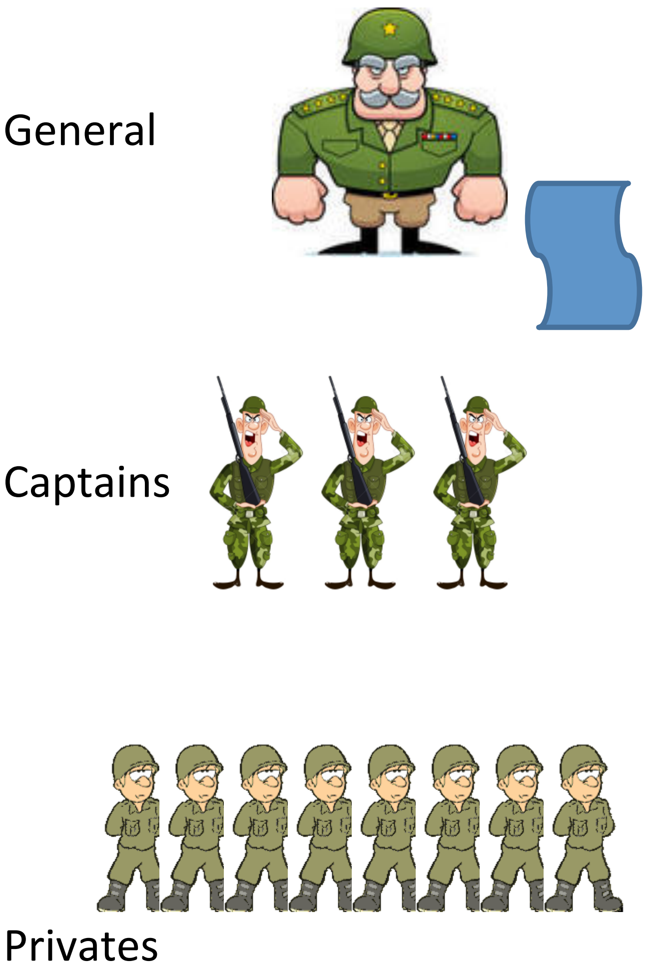
Kenneth J. Biba in 1975

Why no Read Down?



- A higher integrity object may be modified based on a lower integrity document

Example



Read Up

- A document from the general should be read by all

No Read Down

- A private's document should not affect the General's decisions