

OS Security Advanced

CS642:

Computer Security



Learning Goals

- Delegation
- Time-of-check-to-time-of-use
- Confinement
- Multi-level security

Delegation

- Goal: give another program /user ability to act on a subset of your objects
 - Subgoal 1: restrict set of objects
 - Subgoal 2: restrict set of actions
- Unix approach:
 - `setuid()` + File descriptor **inheritance**
 - `setuid` programs
- Windows approach: **impersonation**

UNIX Process permissions

- Process (normally) runs with identity of user that invoked process

A screenshot of a terminal window. The title bar shows 'rist@seclab-laptop1.local: ~/work — passwd — 80x24'. The terminal content shows the user 'rist' at 'seclab-laptop1' in the directory '~/work' running the 'passwd' command. The output indicates the password is being changed for 'rist' and prompts for the 'Old Password' with a masked input field.

```
rist@seclab-laptop1.local: ~/work — passwd — 80x24
rist@seclab-laptop1:~/work$ passwd
Changing password for rist.
Old Password: [masked]
```

/etc/shadow is owned by root

Users shouldn't be able to write to it generally

Privilege Elevation

- How can a user access a resource they don't have permissions to?
 - Option 1: Make a system call, let OS kernel do it (it can do anything!)
 - Option 2: Invoke a program that has more permissions
 - Windows: send a message to a running process with more privilege
 - Linux: setuid programs

Process permissions continued

UID 0 is root

Real user ID (RUID) --

same as UID of parent (who started process)

Effective user ID (EUID) --

from set user ID bit of file being executed or due to sys call

Saved user ID (SUID) --

place to save the previous UID if one temporarily changes it

Also SGID, EGID, etc..

Executable files have 3 setuid bits

- Setuid bit – set EUID of process to owner's ID
- Setgid bit – set EGID of process to group's ID
- sticky bit:
 - 0 means user with write on directory can rename/remove file
 - 1 means only file owner, directory owner, root can do so

So passwd is a setuid program

program runs at permission level of owner, not user that runs it



rist@seclab-laptop1.local: /usr/bin — bash — 80x24

-r-xr-xr-x	1	root	wheel	50512	Feb 10	2011	yes
-r-xr-xr-x	1	root	wheel	50832	Feb 10	2011	ypcat
-r-xr-xr-x	1	root	wheel	50864	Feb 10	2011	ypmatch
-r-xr-xr-x	1	root	wheel	55344	Feb 10	2011	ypwhich
-rwxr-xr-x	2	root	wheel	146976	Feb 10	2011	zcat
-rwxr-xr-x	1	root	wheel	71	Feb 10	2011	zcmp
-rwxr-xr-x	1	root	wheel	4422	Feb 10	2011	zdiff
-rwxr-xr-x	1	root	wheel	66	Feb 10	2011	zegrep
-rwxr-xr-x	1	root	wheel	66	Feb 10	2011	zfgrep
-rwxr-xr-x	1	root	wheel	2017	Feb 10	2011	zforce
-rwxr-xr-x	1	root	wheel	4894	Feb 10	2011	zgrep
-rwxr-xr-x	1	root	wheel	359968	Feb 10	2011	zip
-rwxr-xr-x	1	root	wheel	168432	Feb 10	2011	zipcloak
-rwxr-xr-x	1	root	wheel	1188	Feb 10	2011	zipgrep
-rwxr-xr-x	2	root	wheel	265392	Feb 10	2011	zipinfo
-rwxr-xr-x	1	root	wheel	155440	Feb 10	2011	zipnote
-rwxr-xr-x	1	root	wheel	159632	Feb 10	2011	zipsplit
-rwxr-xr-x	1	root	wheel	1735	Feb 10	2011	zless
-rwxr-xr-x	1	root	wheel	2441	Feb 10	2011	zmore
-rwxr-xr-x	1	root	wheel	4954	Feb 10	2011	znew
-r-xr-xr-x	1	root	wheel	63424	Apr 29	17:30	zprint

rist@seclab-laptop1:/usr/bin\$ ls -al passwd

-r-sr-xr-x 1 root wheel 111968 Apr 29 17:30 passwd

rist@seclab-laptop1:/usr/bin\$

seteuid system call

seteuid can:

- go to SUID or RUID always
- any ID if EUID is 0

```
uid = getuid();
```

```
eid = seteuid();
```

```
seteuid(uid);    // Drop privileges
```

```
...
```

```
seteuid(eid);    // Raise privileges
```

```
file = fopen( "/etc/shadow", "w" );
```

```
...
```

```
seteuid(uid);    // drop privileges
```

Setuid allows necessarily privilege escalation but...

- Source of many privilege escalation vulnerabilities

Control-flow hijacking vulnerability (future lecture) in local setuid program gives privilege escalation

Race conditions

Setuid() best practices

- A setuid program doesn't need privileged EUID all the time
 - If compromise occurs while program is unprivileged, damage is limited
- Operate with “least privilege”:
 - Drop privilege (immediately!) at start of execution
 - Raise privilege temporarily when needed

SetUID: File descriptor inheritance

- Open privileged file
- Setuid() to user
- Exec() user program
- User program has access to open privileged file
 - Good or bad...

```
main (int argc, char **argv)
{
    int fd;
    struct stat st;
    if ((fd = open ("/etc/shadow",O_RDWR)) == -1) {
        perror ("open:");
        exit (-1);
    }
    // Drop privileges
    setreuid (getuid(),getuid());
    setregid (getgid(),getgid());
    if (fork() == 0) { system (argv[1]); exit(0); }
    printf ("bye, bye!!!\n");
    close (fd);
}
```

Preventing Inheritance

- Solution: close file descriptors on exec()
 - `fcntl(fd,FD_CLOEXEC);`

Windows Delegation

- Windows has no `setuid()`
- Instead: authenticated IPC + impersonation
 - Process has a “token” with userID + group IDs
 - Thread optionally has a token
 - Like effective UID/GID in Linux
- Use: run privileged process
 - Forward client IDs to server
 - Server temporarily acts as client (`ImpersonateClient`) during calls

Windows example

- Run privileged, switch to client token when accessing client resources
- Example: print a client file

```
// hToken is client token
if(!ImpersonateLoggedOnUser(hToken))
    // failed
    exit(-1);
}
// access data as user
HFILE fh = OpenFile(file_to_print);
if (HFILE != HFILE_ERROR)
    // act on file
// Terminates the impersonation of a client.
RevertToSelf();
```

Time-of-Check-to-Time-of-Use Bugs

```
Withdraw(account Acct,  
          int Amount) {  
    int b = get_balance(Acct);  
    if (b < Amount) abort;  
  
    b = b - Amount;  
    set_balance(b);  
}
```

```
Withdraw(MikeAcct,  
        $1000)
```


Bank TOCTTOU

- Example: debit card paying for rental car - -why do they charge in advance?

```
procedure withdrawal(w)
  // contact central server to get balance
  1. let b := balance

  2. if b < w, abort
  Balance could have decreased at this point due to another action
  // contact server to set balance
  3. set balance := b - w

  4. dispense $w to user
```

TOCTTOU = Time of Check To Time of Use

File System Races

Basic problem: **race condition** between code and other accesses to data

```
if( access("/tmp/myfile", R_OK) != 0 ) {  
    exit(-1);  
}  
  
file = open( "/tmp/myfile", "r" );  
read( file, buf, 100 );  
close( file );  
print( "%s\n", buf );
```

Where is the bug?

Say program is setuid root:
access checks RUID, but open only checks EUID

`access("/tmp/myfile", R_OK)`



`ln -s /home/root/.ssh/id_rsa /tmp/myfile`


`open("/tmp/myfile", "r");`

`print("%s\n", buf);`

Prints out the root's
secret key...

Better code

```
euid = geteuid();  
ruid = getuid();  
seteuid(ruid);           // drop privileges  
file = open( "/tmp/myfile", "r" );  
read( file, buf, 100 );  
close( file );  
print( "%s\n", buf );
```



No race!

Confinement

- **Context:** running program from untrusted owner on sensitive data
- **Problem:** any program, if confined, will be unable to leak data
 - Store data in memory, wait for next call from owner
 - Write data to file in owner's home directory
 - Write data to temporary file accessible by owner
 - The service may send a message to a process controlled by its owner using IPC.

More generally

- Running a program as a user gives program full capabilities of user
 - Access to all user's resources (e.g., files)
 - User's access to network
- Question: how can we restrict capabilities of a process?
 - Limit what data is available
 - Limit what it can do with data

Solution 1: Restrict privileges

- Problem solved: program has full access of user
- Example: browser plugin
- Solution: *restricted tokens* (Windows)
 - Add additional group IDs to token = restricted IDs
 - On access check, only grant access if:
 - `Access_ok(user) && access_ok(RestrictedIDs)`
 - Example: create restricted ID *chrome-browser*, grant access to browser profile **only**

Solution 2: system call checks

- Problem solved: program has full access of user
- Solution: add extra code on all system calls for extra checks
- How: `ptrace()` invokes external helper process on every syscall
 - Helper verifies arguments.
 - Policies:
 - Disallow some calls (e.g., `fork/exec`)
 - Limit paths (e.g., `open/create`)
 - Limit network endpoints (e.g., `bind, sendto, recvfrom`)

Covert Channels

- Not all information leaked explicitly
 - information may be encoded in a billing statement: # of CPU seconds used == data value
 - Vary ratio of compute to I/O leaks to process than can observe performance
 - File locks that prevent files from being open for writing and reading at the same time can leak data if it is merely allowed to read files which can be written by its owner.

Sending data through file locks

```
set(file,value) {  
    if (value)  
        while (open (file) == ERROR_OPEN))  
            ;  
    else close(file);  
}
```

```
Bool value(file) {  
    if (open(file) == ERROR_OPEN)  
        return FALSE;  
    else return TRUE;  
}
```

Files data, sendclock, recvclock;

Sender(bit):

```
    set(data, bit);  
    set(sendclock, true);
```

Receiver:

```
    while (!value(sendclock) ;  
    bit = value(data);  
    set(recvclock, rue);
```

sender:

```
    while (!value(recvclock)) ;  
    set(sendclock, false);
```

Receiver:

```
    while (value(sendclock)) ;  
    set(recvclock,false);
```

Discretionary Access Control

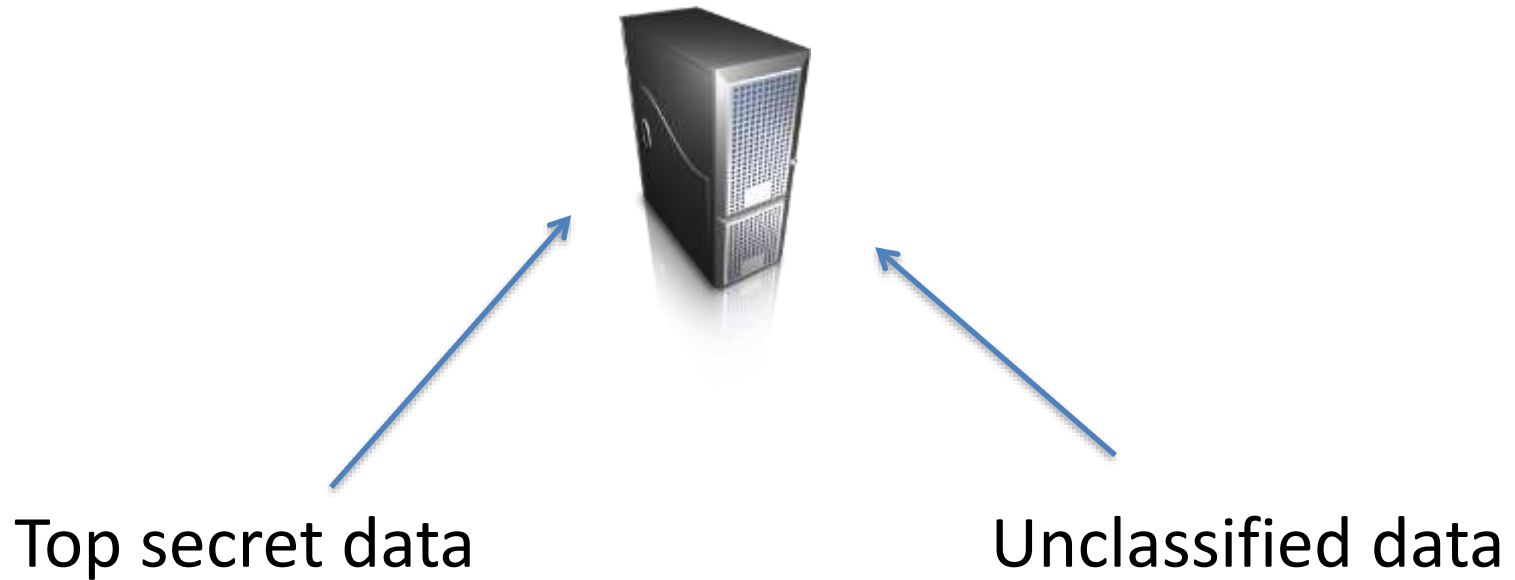
- Unix and Windows use *discretionary access control*:
 1. Owners can control sharing of data
 2. Subject with access can pass that permission to any other subject
- Example:
 - Student in 642 can create homework solutions, share with all other students
 - Instructor gives HW solutions to TAs, they can share with all students

Mandatory Access Control

- Security policy controlled by administrators
 - Users cannot set policy, decide whom to share with
- Controls what a user/program can do with data after access
- Example:
 - Instructor grants homework access to TAs
 - TAs cannot share homework with students
 - Cannot change ACL, cannot pass a capability
 - Cannot write it to a file readable by students

Multi-level security

- Military and other government entities want to use time-sharing too



Classification levels

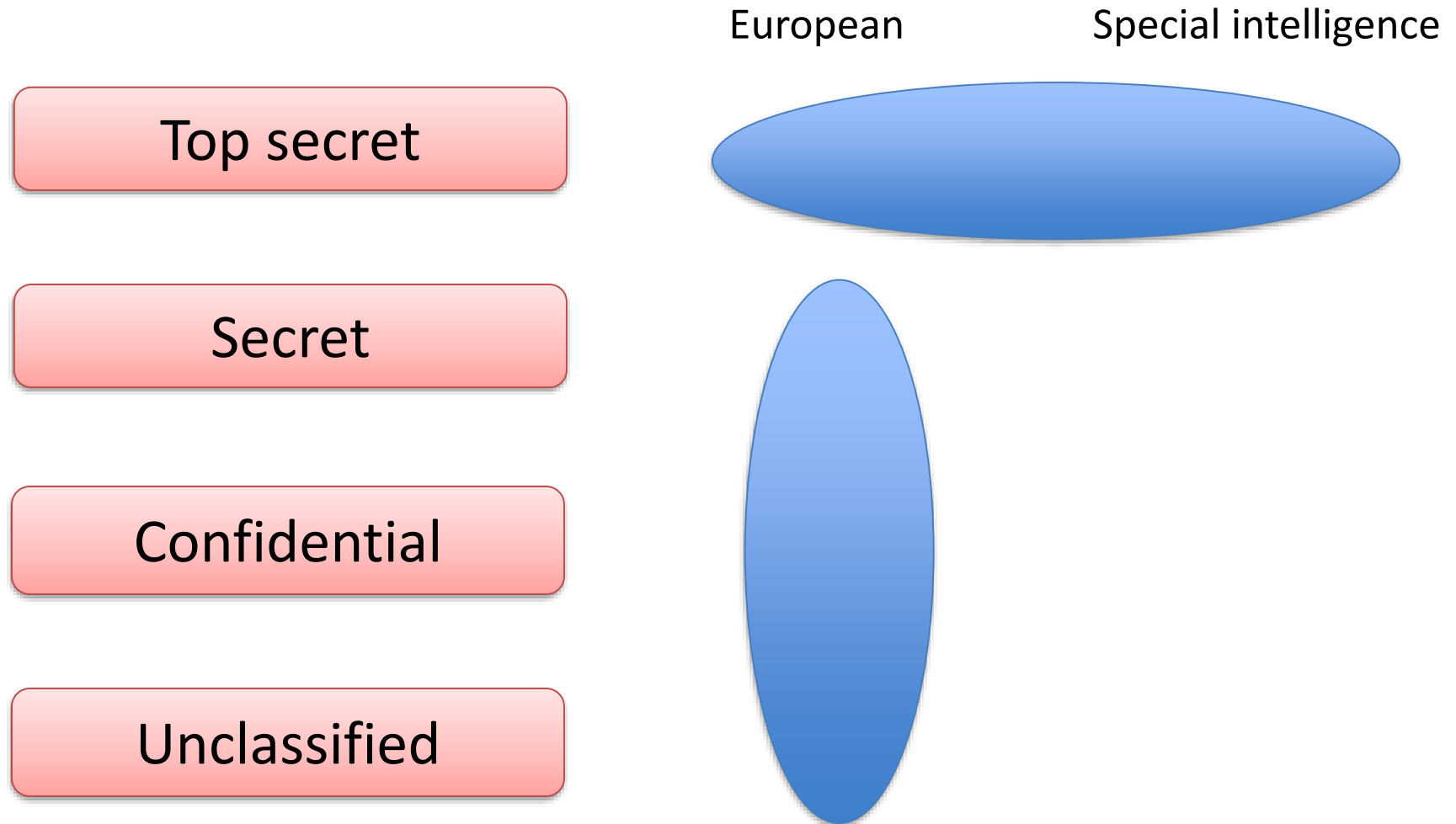
Top secret

Secret

Confidential

Unclassified

Classification levels and compartmentalization



Classification levels and compartmentalization

- Security level (L,C) for process, data, channel
 - L is classification level (Top secret, secret, ...)
 - C is compartment (Europe, Special intelligence...)
- Processes have a *clearance*, objects have a *classification*

Dominance relationship:

$$(L_a, C_a) \leq (L_b, C_b)$$

$L_a < L_b$ (L_1 “less secret” than L_b)

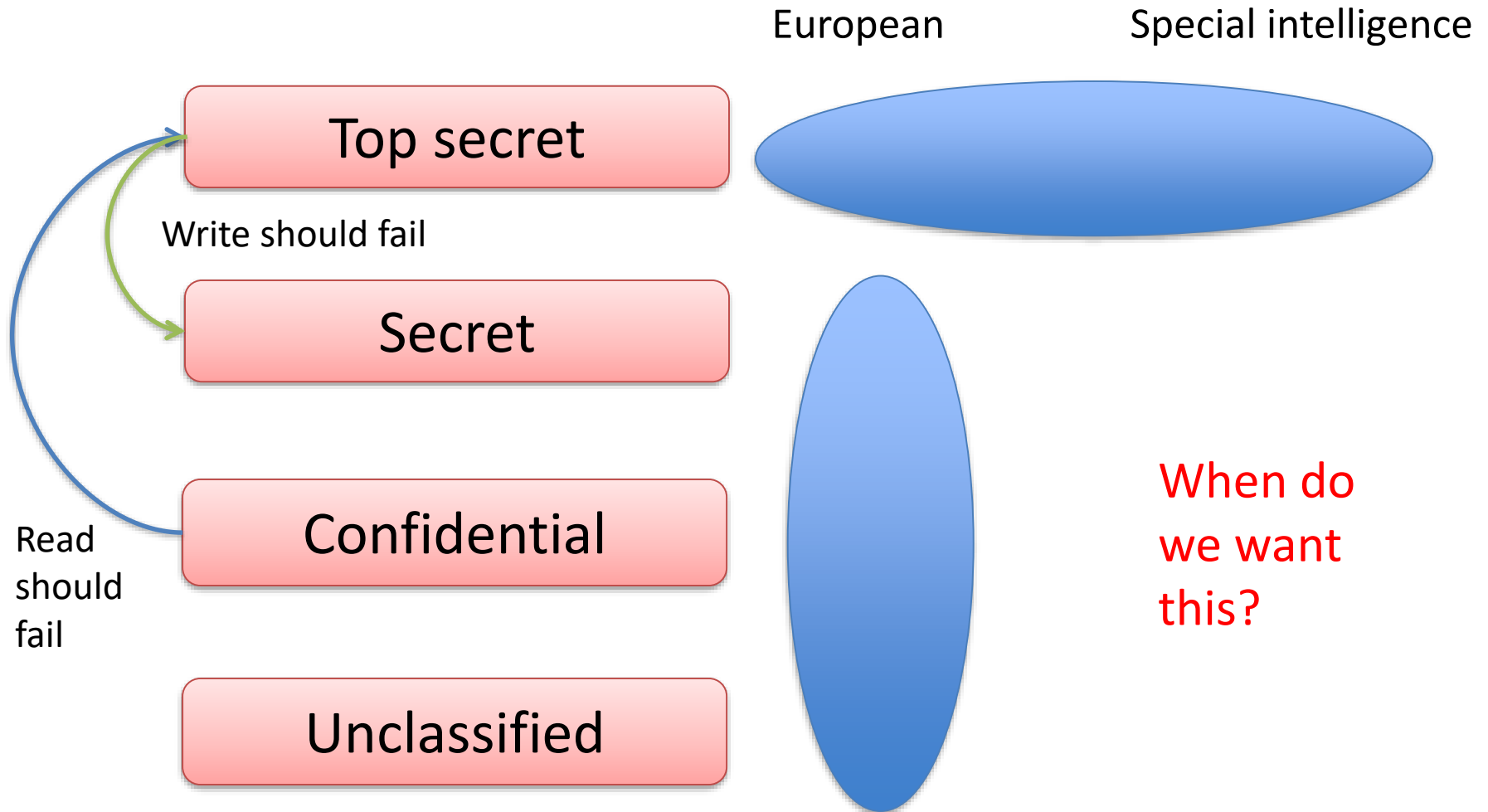
C_a subset of C_b

Example:

$$(\text{Secret}, \{\text{European}\}) \leq (\text{Top Secret}, \{\text{European}, \text{Special Intel}\})$$

Bell-LaPadula Confidentiality Model

“no reads up”, “no writes down”



Bell-LaPadula Confidentiality Model

“no reads up”, “no writes down”

Simple security condition

User with (L_a, C_a) can read file with (L_b, C_b) if?

~~$(L_a, C_a) \leq (L_b, C_b)$~~ or $(L_a, C_a) \geq (L_b, C_b)$

*-property

User with (L_a, C_a) can write file with (L_b, C_b) if?

$(L_a, C_a) \leq (L_b, C_b)$ or ~~$(L_a, C_a) \geq (L_b, C_b)$~~



Say we have just Bell-Lapadula in effect... what could go wrong?

Biba integrity model

“no read down”, “no writes up”

European

Special intelligence

Top secret

Read should fail

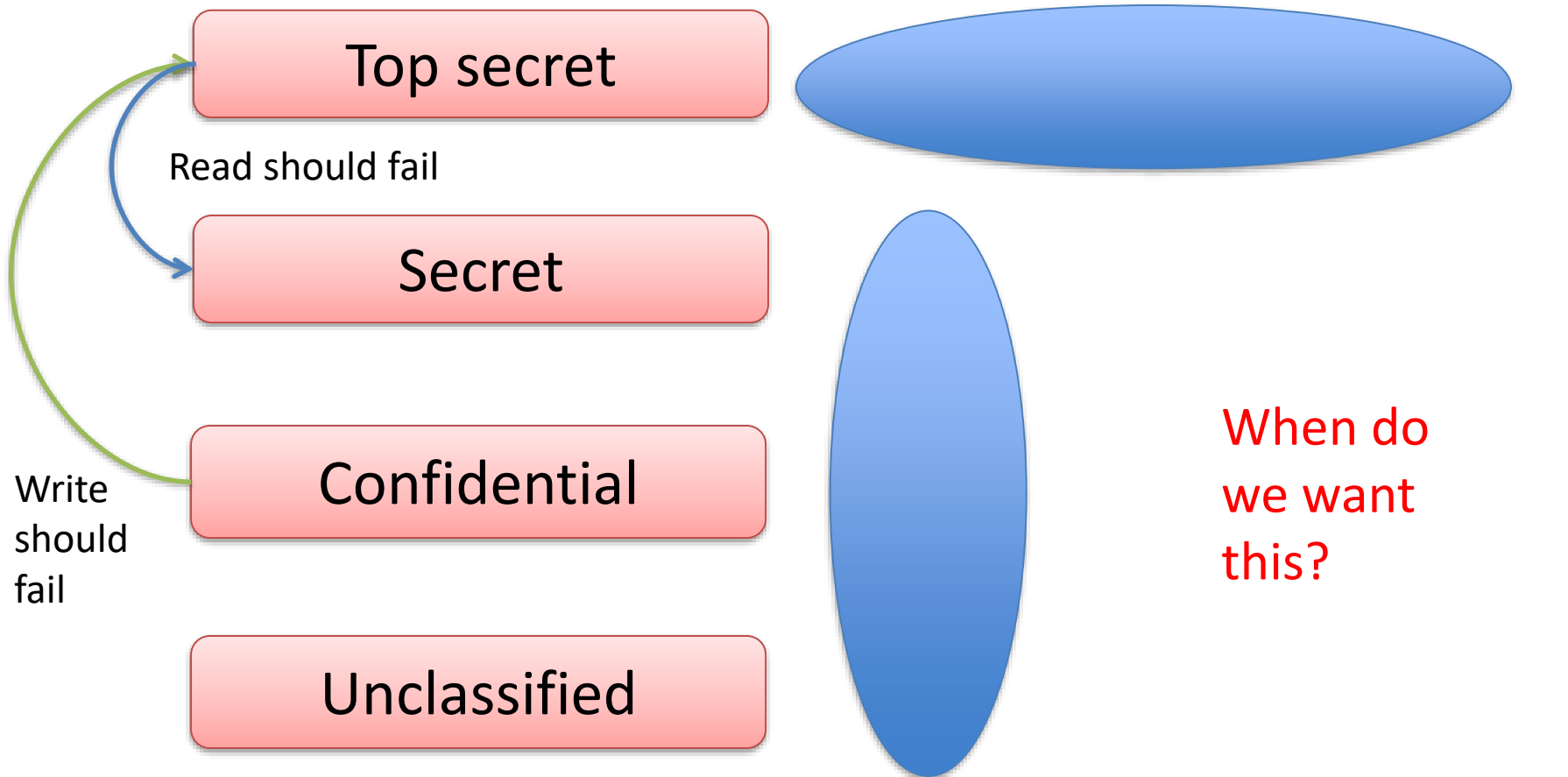
Secret

Confidential

Write
should
fail

Unclassified

When do
we want
this?



Biba integrity model

“no read down”, “no writes up”

Simple integrity condition

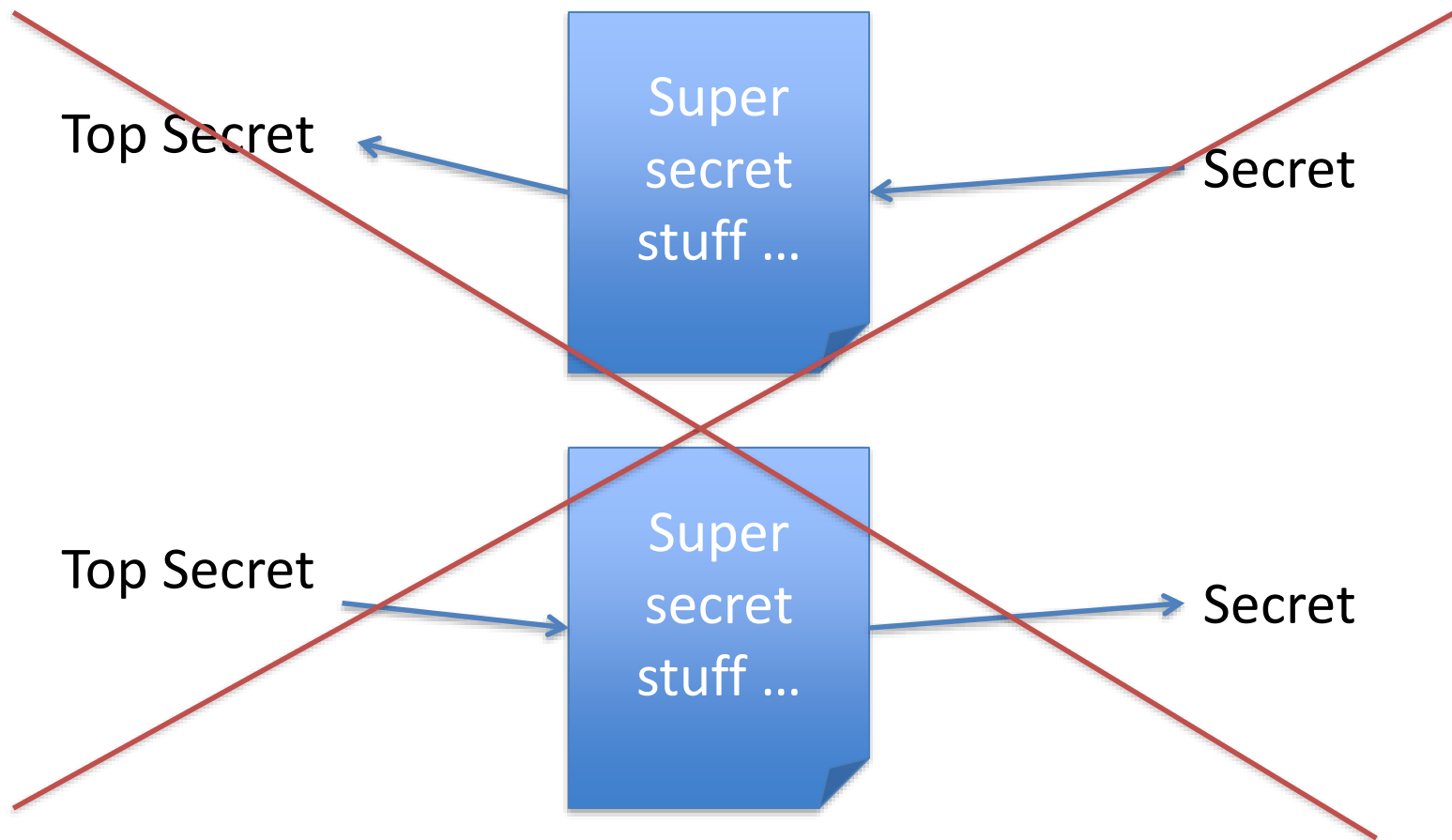
User with (L_a, C_a) can read file with (L_b, C_b) if?

$(L_a, C_a) \leq (L_b, C_b)$ or ~~$(L_a, C_a) > (L_b, C_b)$~~

*-property

User with (L_a, C_a) can write file with (L_b, C_b) if

~~$(L_a, C_a) > (L_b, C_b)$~~ or $(L_a, C_a) \geq (L_b, C_b)$



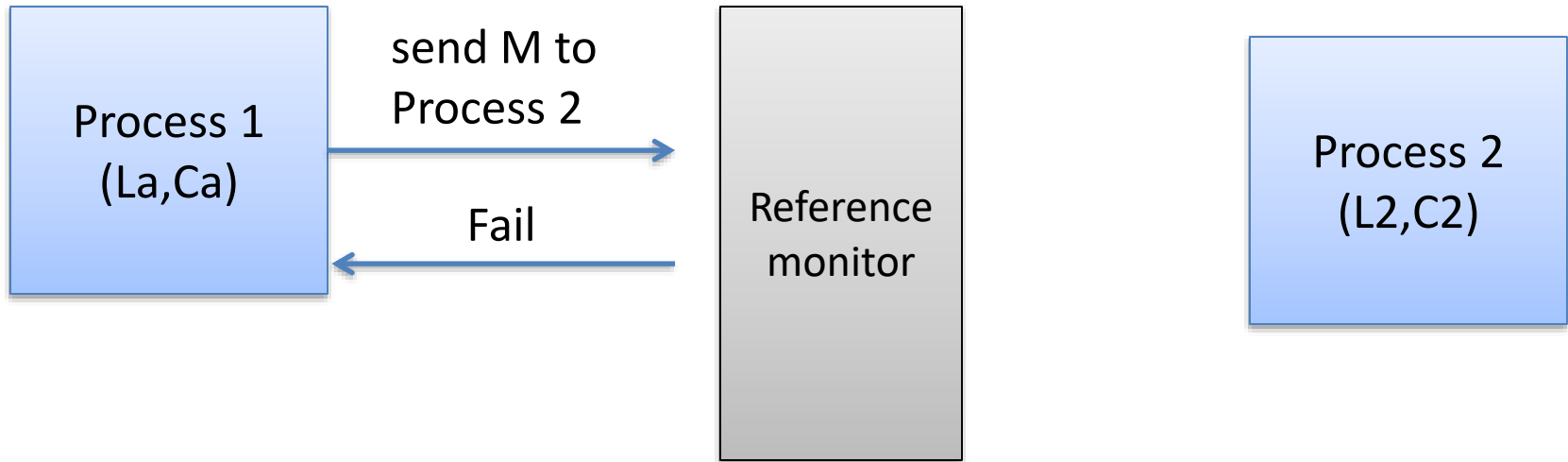
If we combine them... one can only communicate in same classification

Quiz

- You are the admin for www.wisc.edu
 - You want to make sure nobody posts fake snow/cold days
 - Students can't change faculty web pages, etc.
 - What policy do you want for web content?
- You manage the file server storing final exams
 - You want instructors to be able to access exams for all their classes, TAs for just their classes, and students not at all
 - What policy do you want for exams?

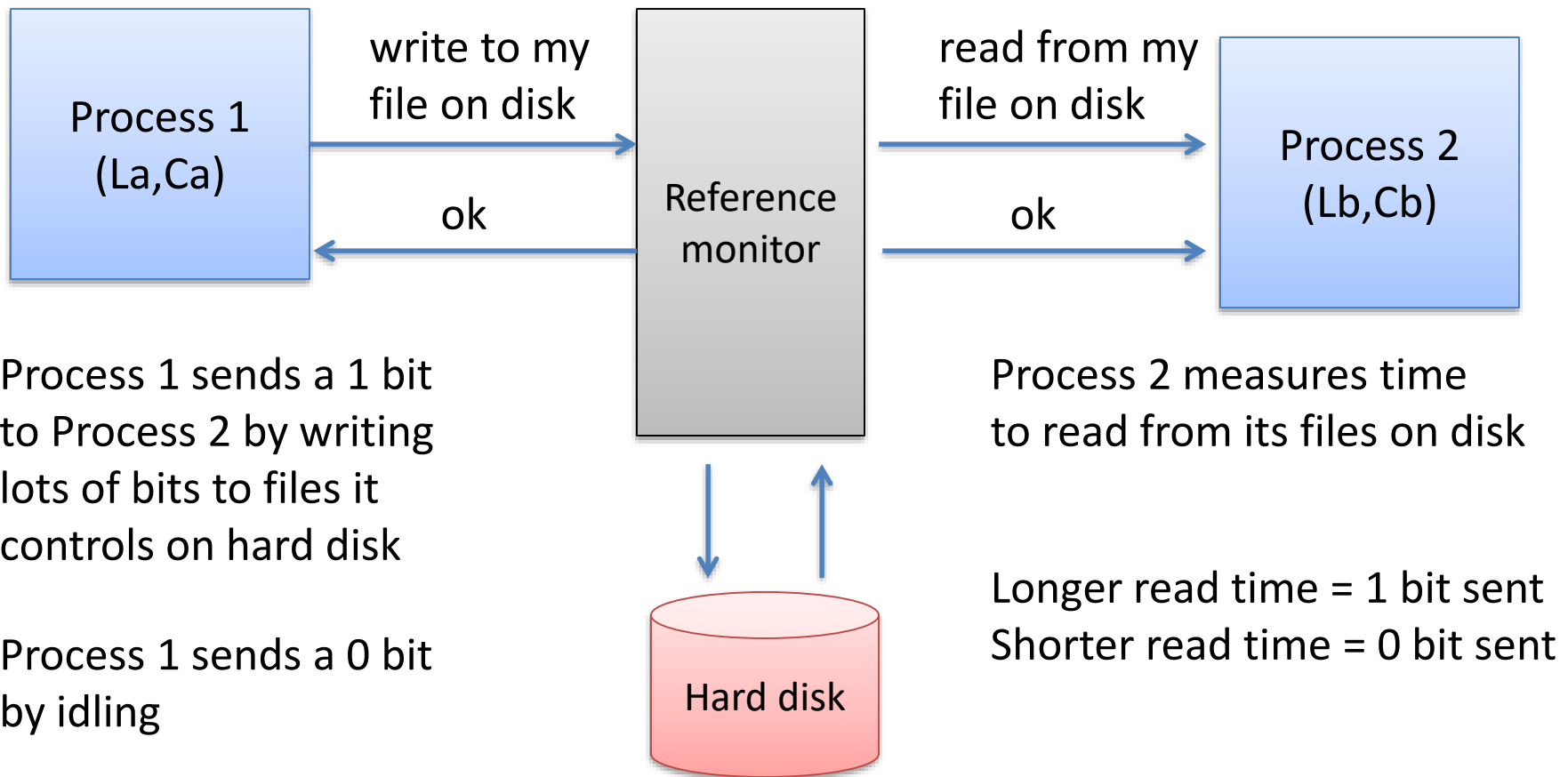
Circumventing access controls: covert channels

$$(L_a, C_a) \geq (L_b, C_b)$$



Circumventing access controls: covert channels

$$(L_a, C_a) \geq (L_b, C_b)$$



Covert channels one reason shared MLS systems unsolved problem



Beyond the Access Matrix policy models

- Decentralized information flow control
- Chinese wall
- Clarke-Wilson integrity model

A good reference is:

Bishop, Computer Security: Art and Science