Security Part 6

Live Anonymous Q&A:
https://tinyurl.com/cs1670feedback

Windows Security

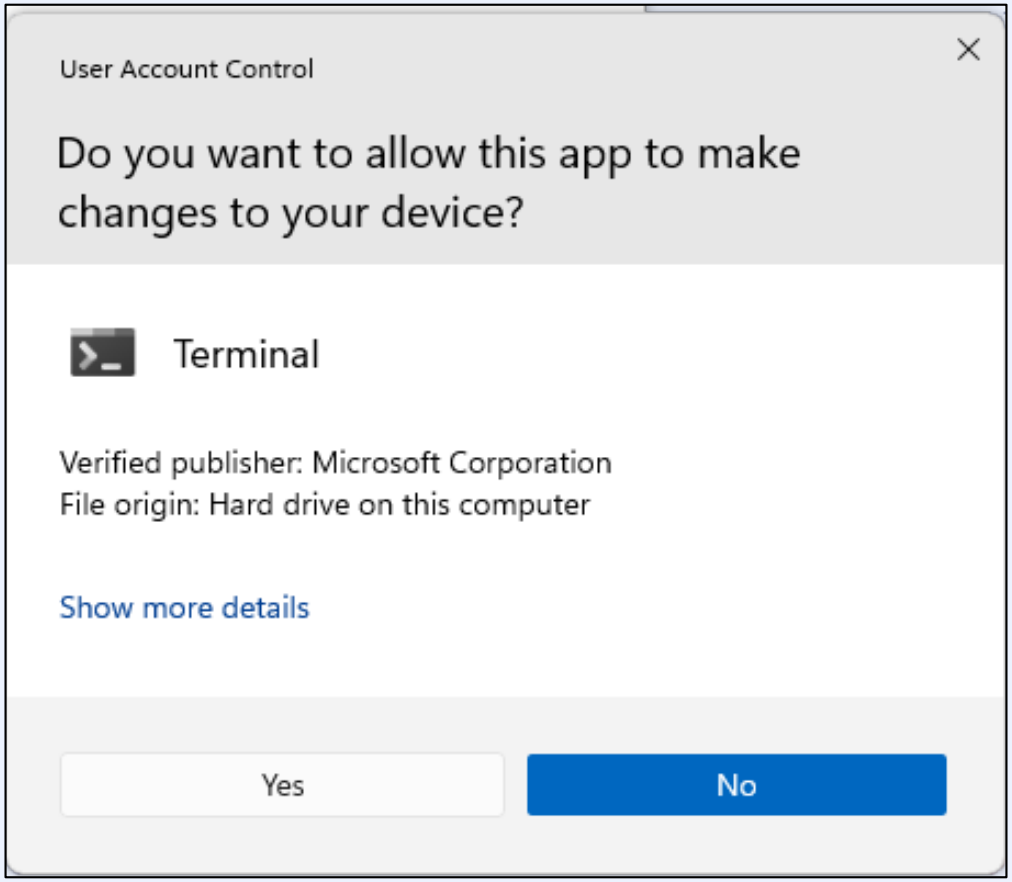
Back to Windows

- Security history
  - DOS and early Windows
    - no concept of logging in
    - no authorization
    - all programs could do everything
  - later Windows
    - good authentication
    - good authorization with ACLs
    - default ACLs are important
      - few understand how ACLs work …
    - many users ran with admin privileges
      - all programs can do everything …

Privileges in Windows

- Properties of accounts
  - administrator ≈ superuser
  - finer breakdown for service applications
- User Account Control (UAC)
  - starting with Vista
  - accounts with administrator privileges have two access tokens
    - one for normal usage
    - another with elevated rights

Windows UAC Example



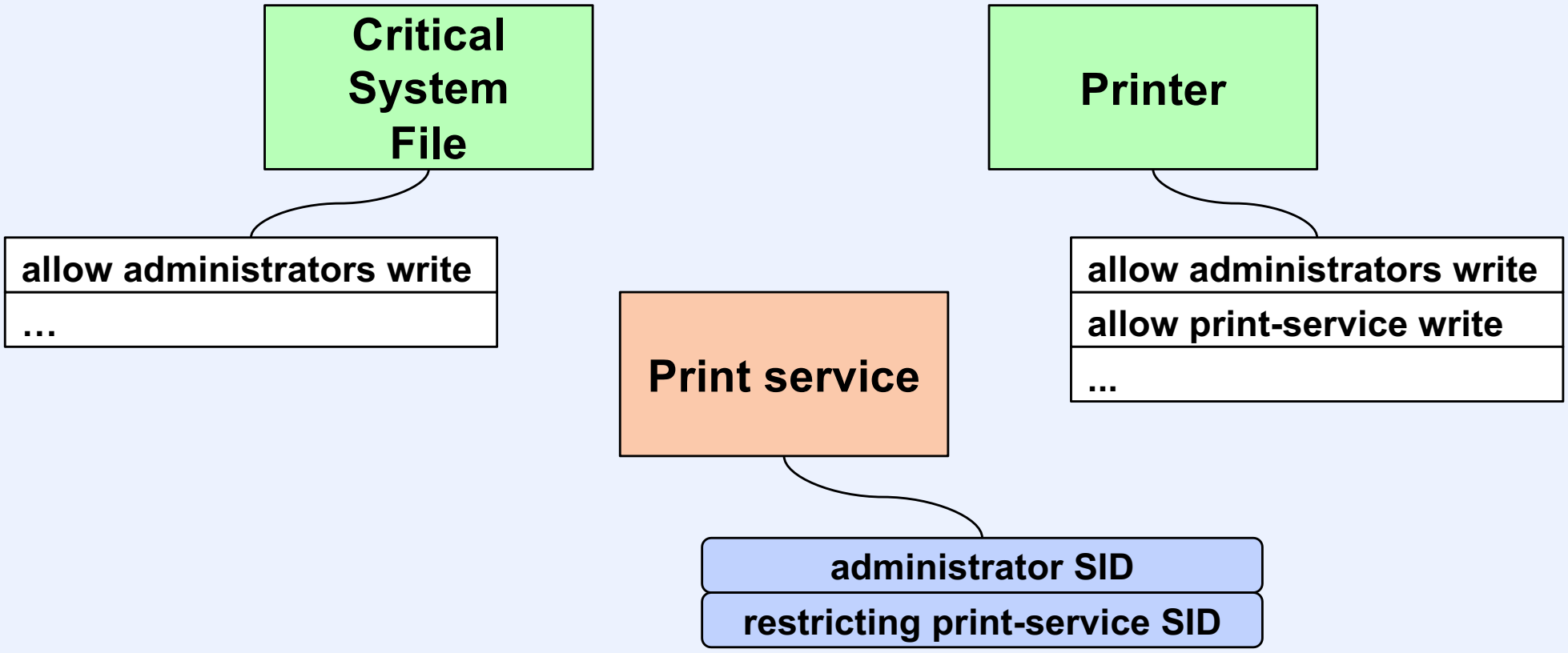
Least Privilege

- Easy answer
  - disable privileges
  - works only if the process has any …
- Another answer
  - restricting SIDs
    - limit what a server can do
    - two passes over ACL for access check
      - first: as previously specified
      - second: using only restricting SIDs

Least Privilege for Servers

- Pre-Vista:
  - services ran in local system account
    - all possible privileges
    - successful attackers “owned” system
    - too complicated to give special account to each service
- Vista and beyond
  - services still run in system account
  - per-service SIDs created
    - used in DACLs to indicate just what service needs
    - marked *restricting* in service token

Example



Not a Quiz

- Why are there two passes made over the ACL?

- Answer: a restricting SID is not an additional access right, but it diminishes what can be done with existing rights
  - one must first show that one has an access right, then check if it has been diminished

Least Privilege for Clients

- Pre Vista
  - no
- Vista and beyond
  - windows integrity mechanism
    - a form of MAC

Print Server

- Client sends request to server
  - print contents of file X
- Server acts on request
  - does client have read permission?
    - server may have (on its own) read access, but client does not
    - server might not have read access, but client does

Unix Solution

- Client execs print-server, passing it file name
  - set-uid-root program
  - it (without races!) checks that client has access to file, then prints it

Windows Solution

- Server process started when system is booted
- Clients send it print requests
  - how does client prove to server it has access?
  - how does server prove to OS that client has said ok?

Impersonation

- Client sends server *impersonation token*
  - subset of its access token
- Server temporarily uses it in place of its own access token

Quiz 1

I've written a print server. You would like to use it to print a file. However, you don't trust me — you're concerned that my print server software might read some of your files that you don't want me to read. My print server uses either the Unix approach (setuid-to-twd) or the Windows approach (you send it an impersonation token) to deal with access control.

a) You have nothing to worry about

b) You have nothing to worry about if it uses the Unix approach

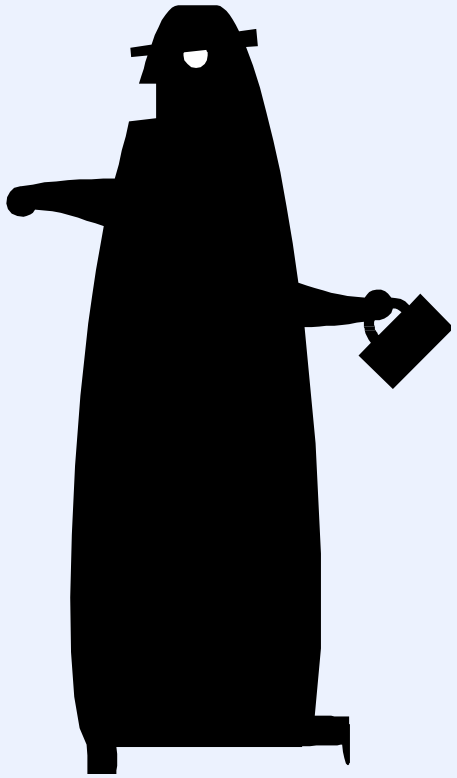
c) You have nothing to worry about if it uses the Windows approach

d) You have a lot to worry about with both

Security Models

Serious Security

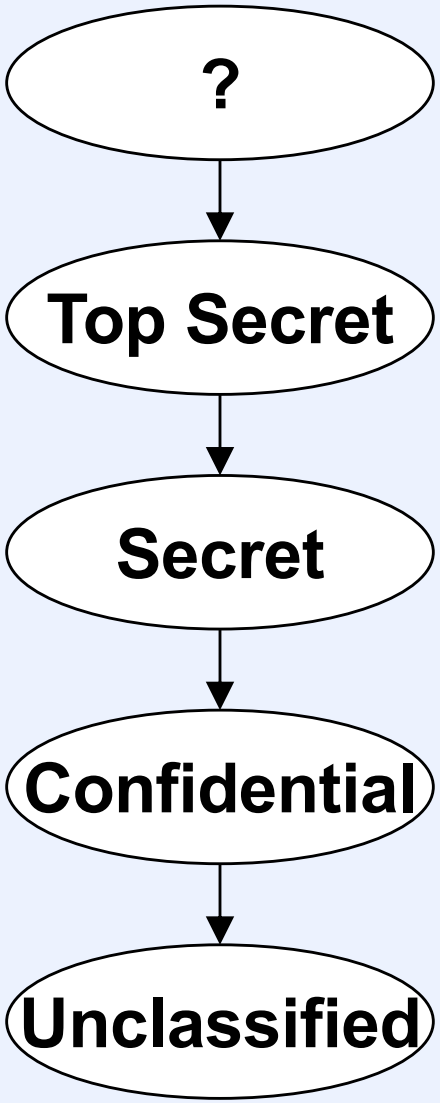
- National defense
- Proprietary information
- Personal privacy



Mandatory vs. Discretionary Access Control

- Discretionary
  - ACLs, capabilities, etc.
    - access is at the discretion of the owner
- Mandatory
  - government/corporate security, etc.
    - access is governed by strict policies

Mandatory Access Control (1)



Mandatory Access Control (2)

- Privacy/confidentiality policies
  - compartmentalization

registrar

dean of the faculty

University-affiliated hospitals

Mandatory Access Control (3)

- Local computer policy
  - web-server
    - may access only designated web-server data
  - administrators
    - may execute only administrative programs
    - (may not execute code supplied by ordinary users)

Bell-LaPadula Model

1) Simple security property — **no-read-up**
   - no subject may read from an object whose classification is higher than the subject's clearance
2) *-property — **no-write-down**
   - no subject may write to an object whose classification is lower than the subject's clearance

Information Black Hole

Attack!

?

Top Secret

Secret

Confidential

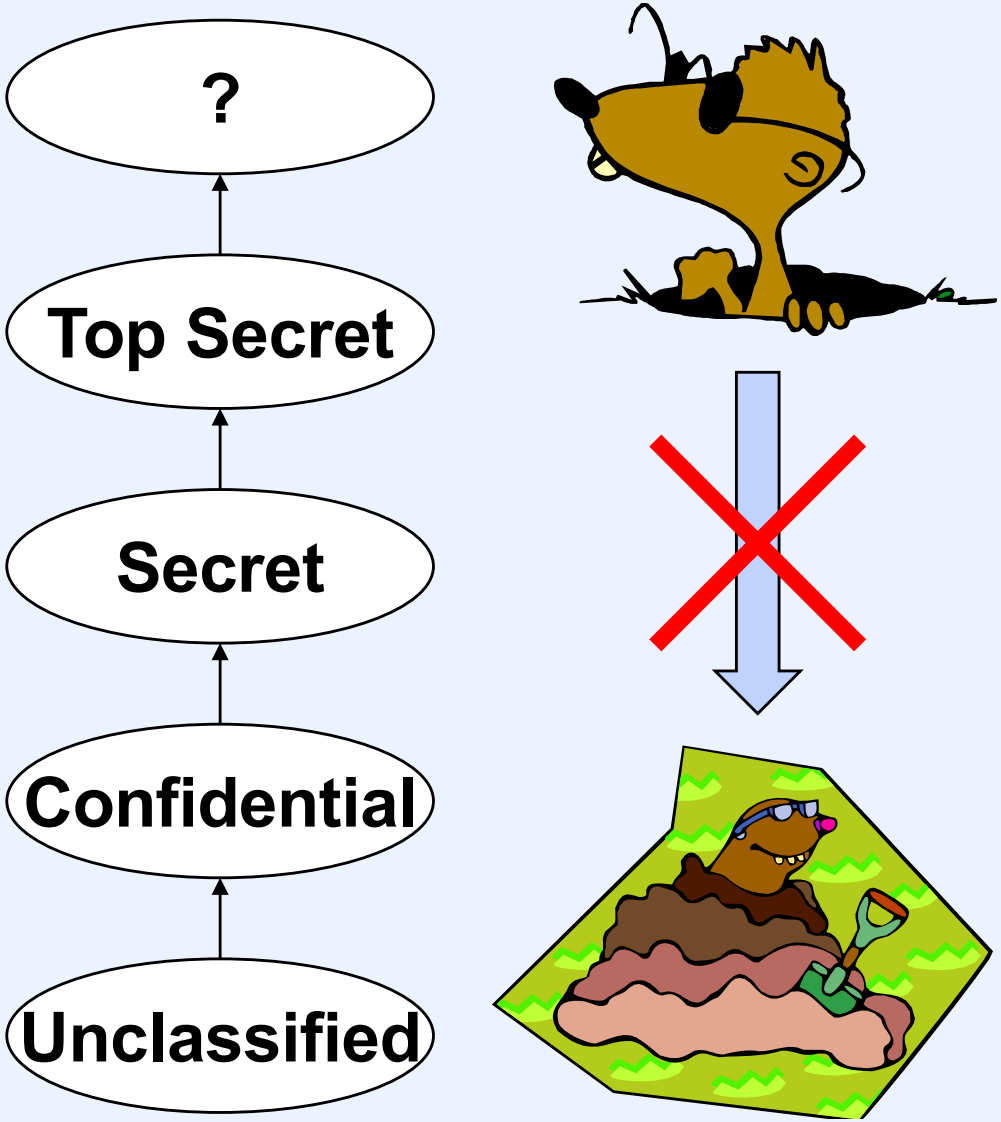
Unclassified

Not cleared for top-secret orders

Managing Confidentiality

- Black-hole avoidance
  - trusted vs. untrusted subjects
  - trusted subjects may write down

Espionage

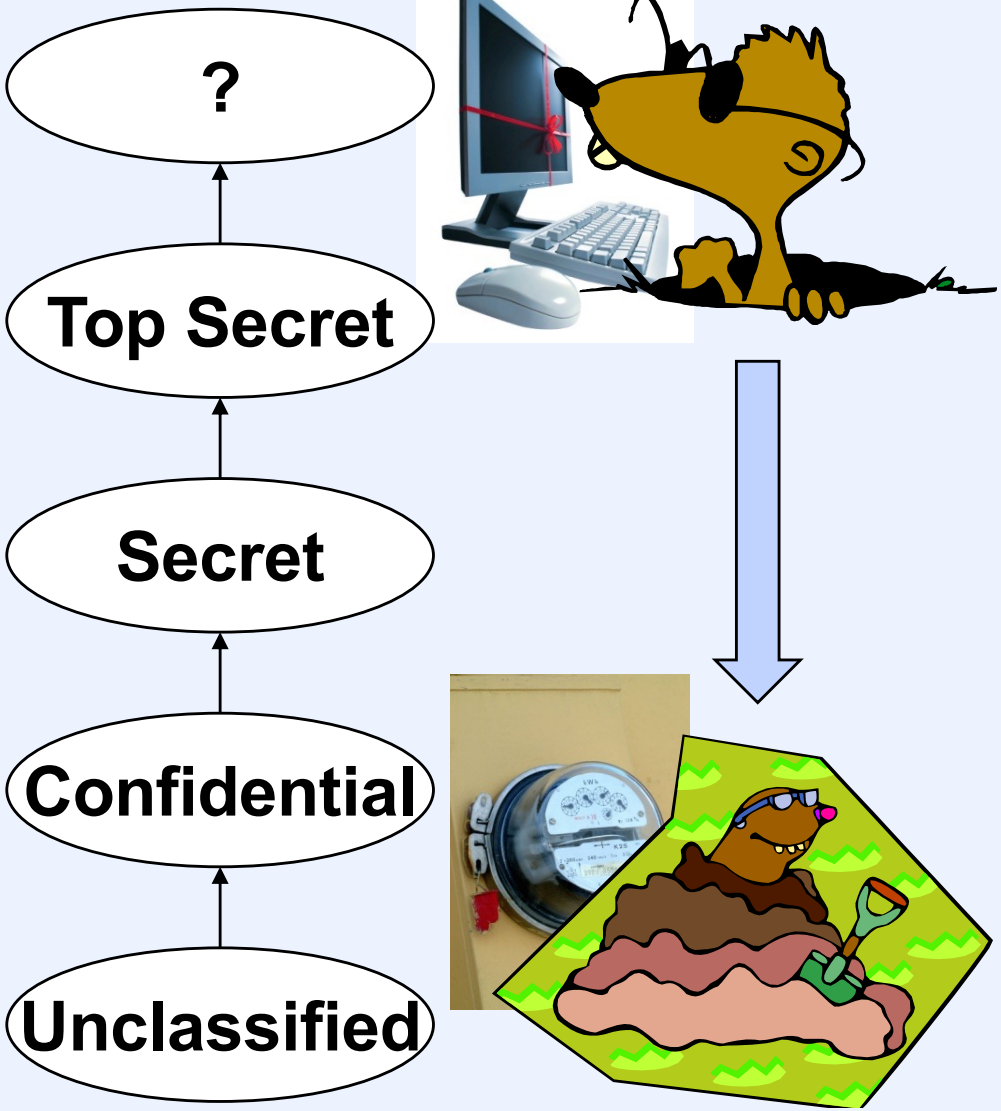


agent X learns of invasion plans

communication not possible

agent Y can send email to spymaster (but doesn't know what to send)

Covert Channels



agent X runs resource-intensive program

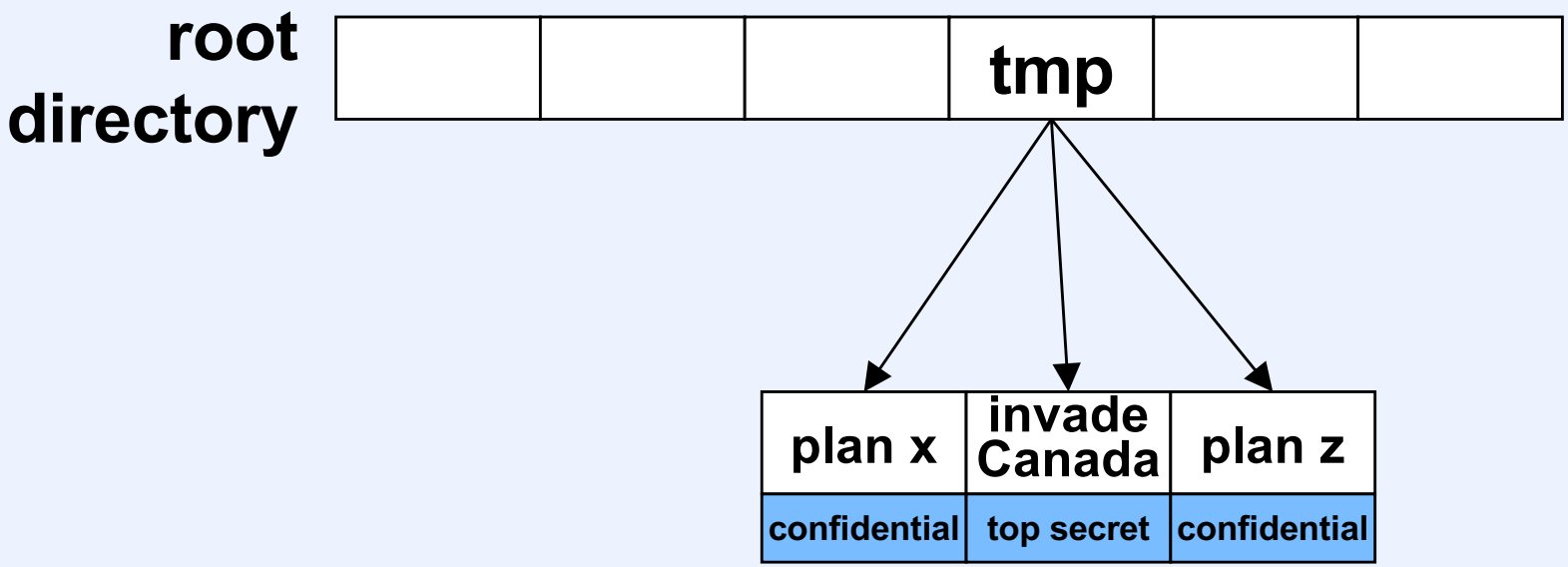
sneaky communication possible

agent Y monitors load sends email to spymaster

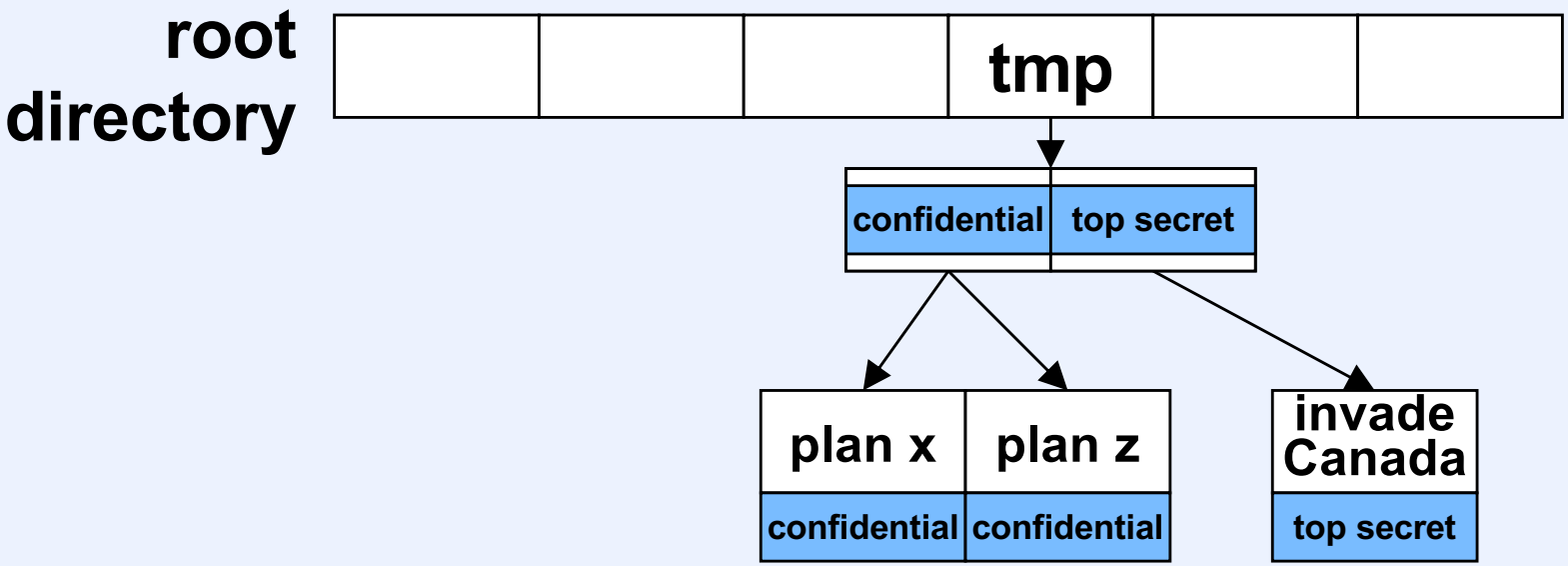
Defense

- Identify all covert channels
  - (good luck …)
- Eliminate them
  - find a suitable scheduler
    - eliminates just one channel

Multi-Level Directories (1)



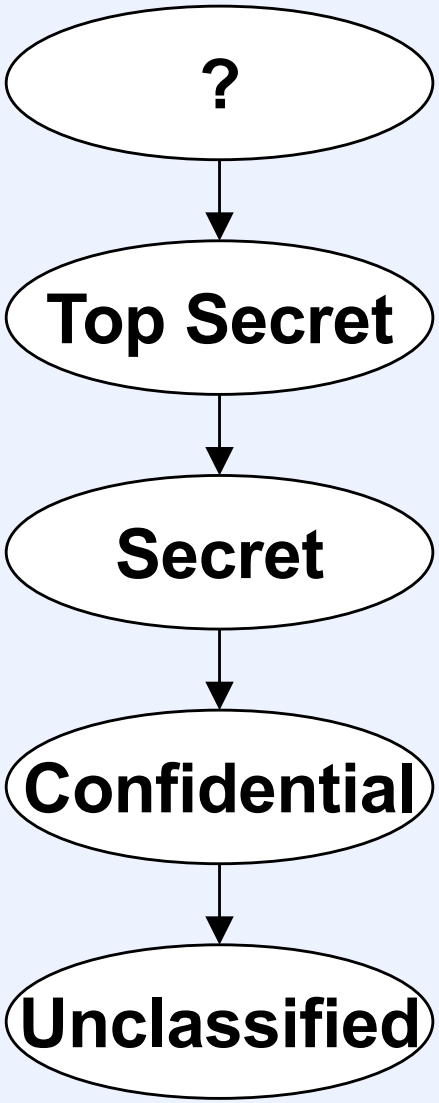
Multi-Level Directories (2)

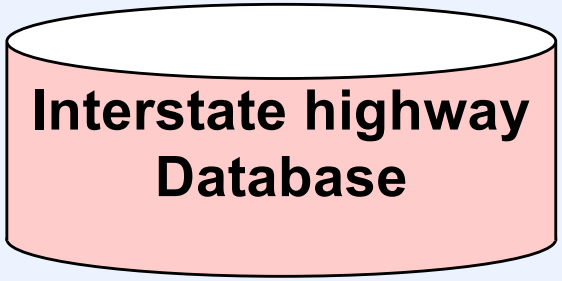


Orange Book

- Evaluation criteria for secure systems
  - D: minimal protection
  - C: discretionary protection
    - C1: discretionary security protection
    - C2: controlled access protection
  - B: mandatory protection
    - B1: labeled security protection
    - B2: structured protection
    - B3: security domains
  - A: verified protection
    - A1: verified design

Integrity





Biba Model

- Integrity is what's important
  - no-write-up
  - no-read-down

Quiz 2

You're concerned about downloading malware to your computer and very much want to prevent it from affecting your computer. Which would be the most appropriate policy to use?

a) no write up

b) no read up

c) no write down

d) no read down

Windows and MAC

- Concerns
  - viruses
  - spyware
  - etc.
- Installation is an integrity concern
- Solution
  - adapt Biba model

Windows Integrity Control

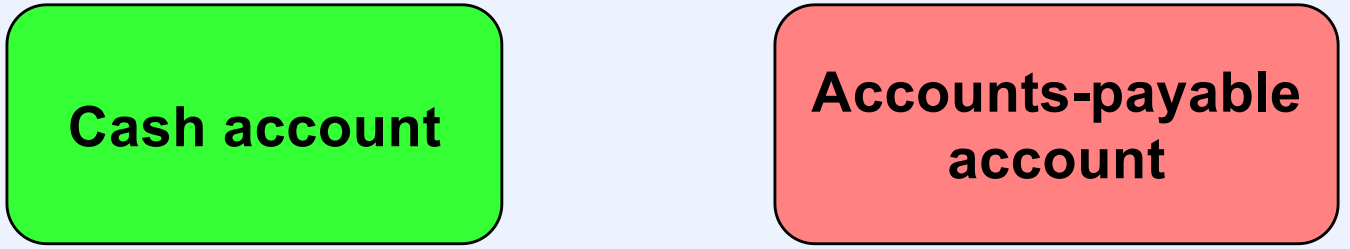
- No-write-up
- All subjects and objects assigned a level
  - untrusted
  - low integrity
    - Internet Explorer/Edge
  - medium integrity
    - default
  - high integrity
  - system integrity
- Object owners may lower integrity levels
- May set *no-read-up* on an object

Industrial-Strength Security

- **Target:**
  - **embezzlers**

Clark-Wilson Model

- Integrity and confidentiality aren't enough
  - there must be control over how data is produced and modified
    - well formed transactions

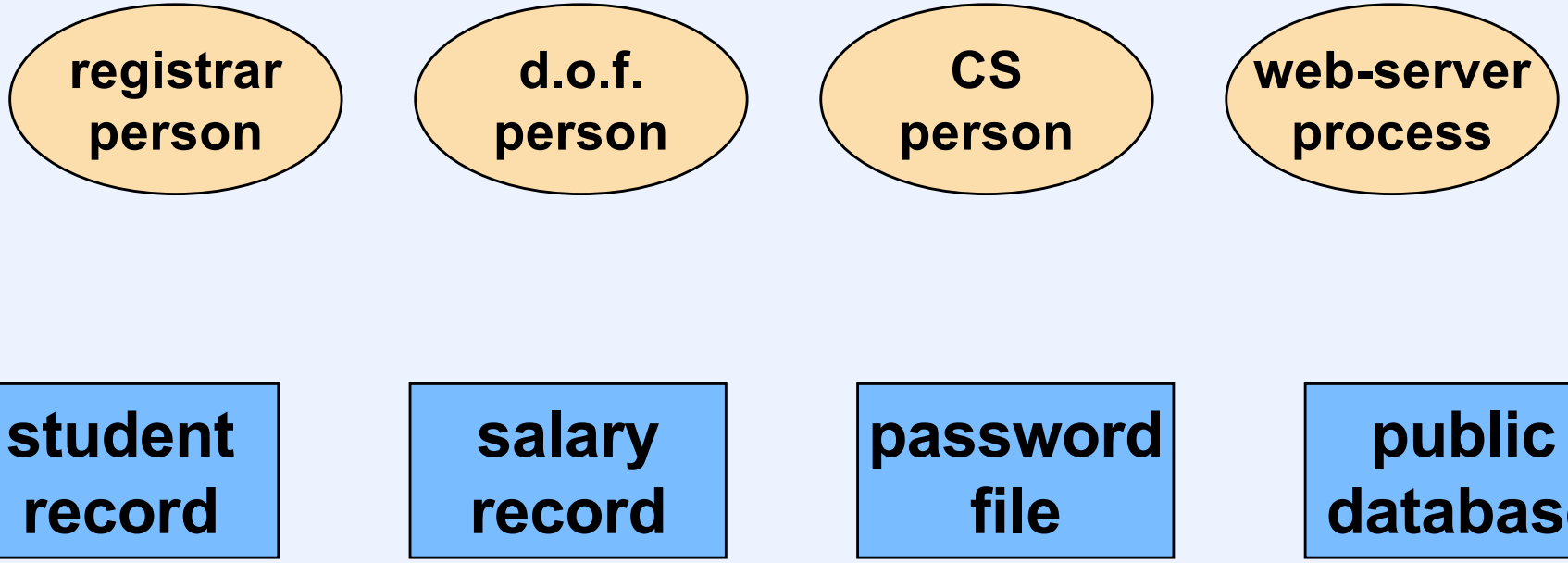


- Separation of duty
  - steps of transaction must involve multiple people

Mandatory Access Control (MAC)

Implementing MAC

- Label subjects and objects
- Security policy makes decisions based on labels and context



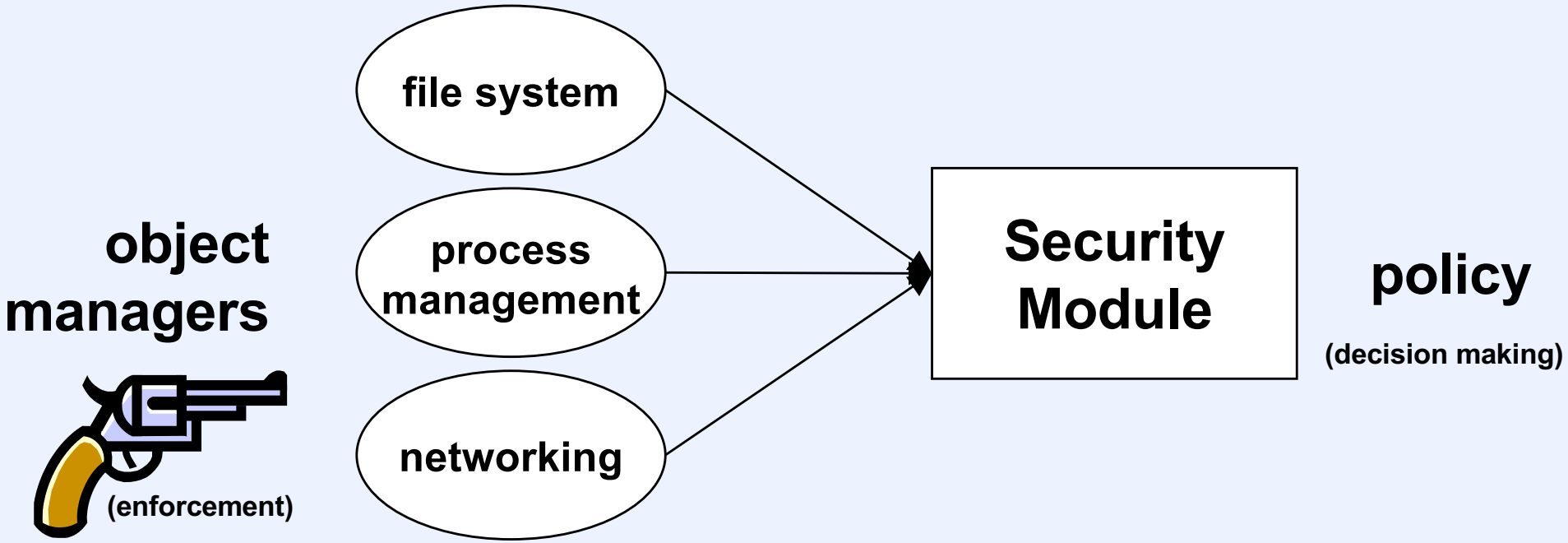
Quiz 3

I have a file that I accidentally set as having rw permission for everyone (0666). You have a process that has opened my file rw. I discover this and immediately change the permissions to 0600 (access only by me). Can your process still read and write the file?

a) It can read and write
b) It can read, but not write
c) It can write, but not read
d) It can do neither

SELinux

- Security-Enhanced Linux
  - MAC-based security
  - labels on all subjects and objects
  - policy-specification language



SELinux Examples (1)

- **Publicly readable files assigned type *public_t***
- **Subjects of normal users run in domain *user_t***
- **/etc/passwd: viewable, but not writable, by all**
- **/etc/shadow: protected**
- **SELinux rules**

```
allow user_t public_t : file read
```

  - **normal users may read public files**

```
allow passwd_t passwd_data_t : file {read write}
```

  - **/etc/shadow is of type *passwd_data_t***
  - **subjects in *passwd_t* domain may read/write /etc/shadow**

SELinux Examples (2)

- **How does a program get into the *passwd_t* domain?**
  - **assume passwd program is of type *passwd_exec_t***

```
allow passwd_t passwd_exec_t : file entrypoint
allow user_t passwd_exec_t : file execute
allow user_t passwd_t : process transition
type_transition user_t passwd_exec_t : process
  passwd_t
```

Quiz 4

We've seen how the setuid feature in Unix is used to allow normal users to change their passwords in /etc/shadow.

a) This approach actually isn't secure, which is among the reasons why SELinux exists

b) The approach is secure and thus SELinux doesn't really add any additional protection to /etc/shadow

c) The approach is secure but there are other potential /etc/shadow-related vulnerabilities that SELinux helps deal with

SELinux Examples (3)

- **Accounting example**
  - **one person requests a purchase order; another approves it**
  - **files containing accounting data are of type *account_data_t***
  - **subjects accessing data are in two domains**
    - ***account_req_t***
    - ***account_approv_t***

```
allow account_req_t account_data_t : file {read
  write}
allow account_approv_t account_data_t : file
  {read write}
```

SELinux Examples (4)

- **Must specify which programs must be used to manipulate accounting data**
  - **requestPO**
    - **used to request a purchase order**
    - **type *account_req_exec_t***
  - **approvePO**
    - **used to approve purchase order**
    - **type *account_approv_exec_t***

```
allow account_req_t account_req_exec_t : file
  entrypoint
allow account_approv_t account_approv_exec_t :
  file entrypoint
```

SELinux Examples (5)

- **Who may run these programs?**

```
allow user_t account_req_t : process transition
allow user_t account_approv_t : process transition
```

Not a Quiz

- Our goal is to make sure that only certain people can request purchase orders, and only certain other people can approve purchase orders
- Do we have the machinery yet to achieve this goal?

SELinux Examples (6)

- **Restrict usage to those users in appropriate roles**

```
role POrequester_r types account_req_t
role POapprover_r types account_approv_t

user mary roles {user_r POrequester_r}
user robert roles {user_r POapprover_r}
allow user_r {POrequester_r POapprover_r}
role_transition user_r account_req_exec_t
  POrequester_r
role_transition user_r account_approv_exec_t
  POapprover_r
```

SELinux Examples (7)

- **Finally …**

```
allow user_t {account_req_exec_t
  account_approv_exec_t} : file execute
```

  - **allow mary and robert to execute programs they need to run**

Off-the-Shelf SELinux

- Strict policy
  - normal users in *user_r* role
  - users allowed to be administrators are in *staff_r* role
    - but may run admin commands only when in *sysadm_r* role
  - policy requires > 20,000 rules
  - tough to live with
- Targeted policy
  - targets only “network-facing” applications
  - everything else in *unconfined_t* domain
  - ~11,000 rules

Capability-Based Systems

Confused-Deputy Problem

- The system has a pay-per-use compiler
  - keeps billing records in file /u/sys/comp/usage
  - puts output in file you provide
    - /u/you/comp.out
- The concept of a pay-per-use compiler annoys you
  - you send it a program to compile
  - you tell it to put your output in /u/sys/comp/usage
  - it does
    - it's confused
    - you win

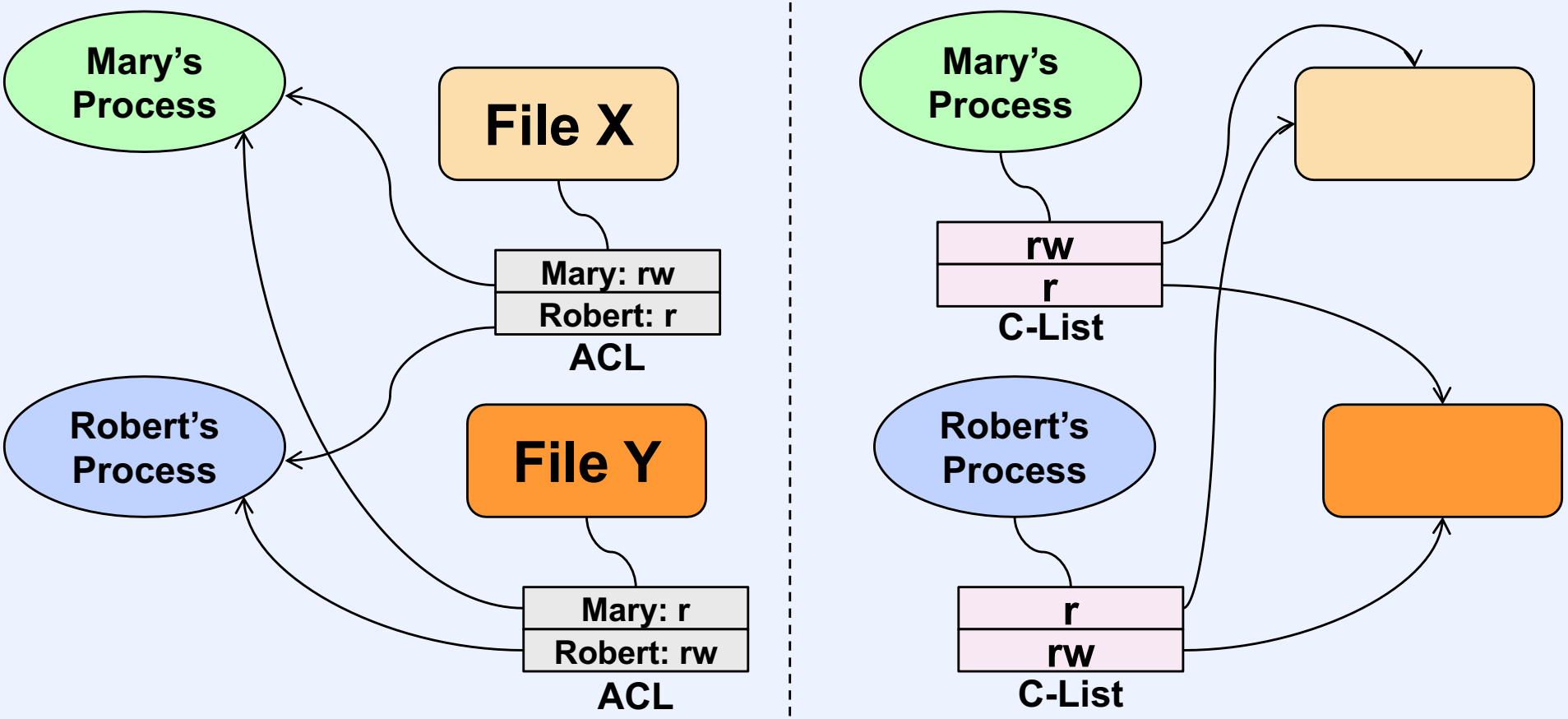
Unix and Windows to the Rescue

- Unix
  - compiler is “su-to-compiler-owner”
- Windows
  - client sends impersonation token to compiler
- Result
  - malicious deputy problem
- Could be solved by passing file descriptors
  - not done
  - should be …

Authority

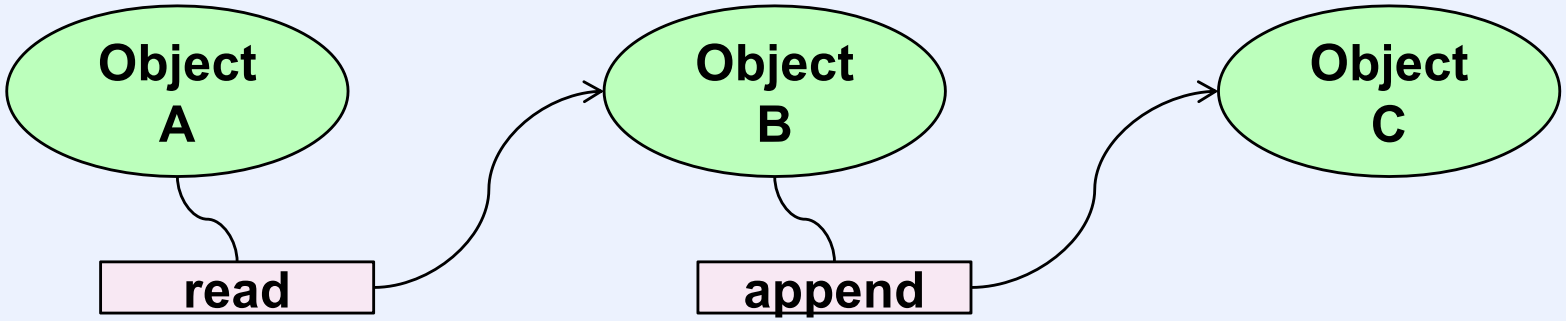
- Pure ACL-based systems
  - authority depends on subject's user and group identities
- Pure capability-based systems
  - authority depends upon capabilities possessed by subject

ACLs vs. C-Lists

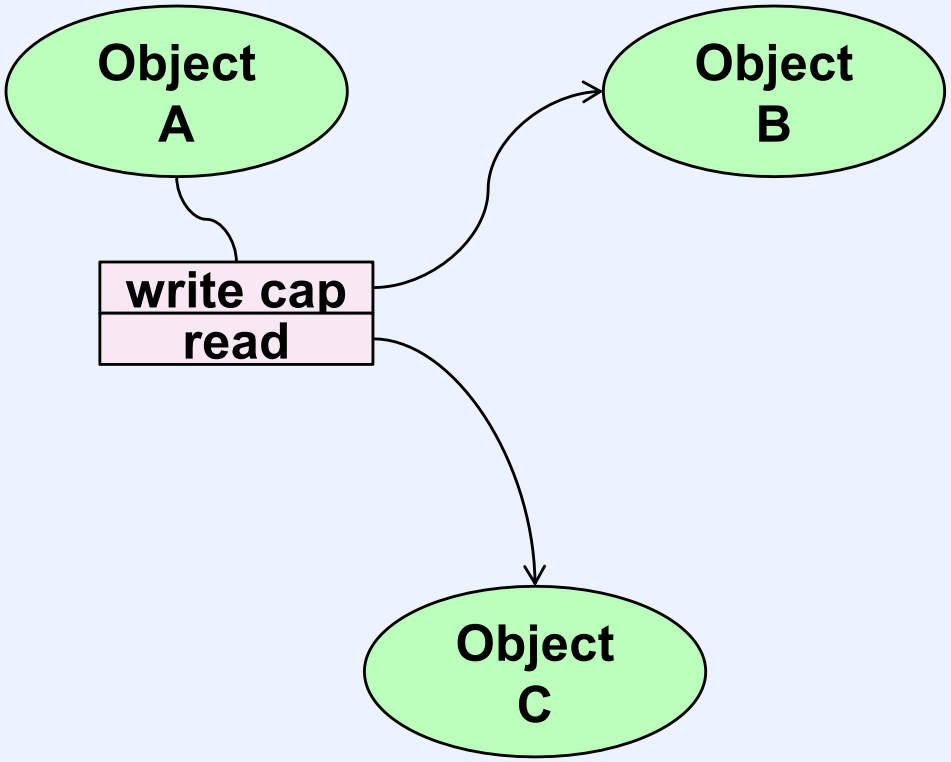


More General View

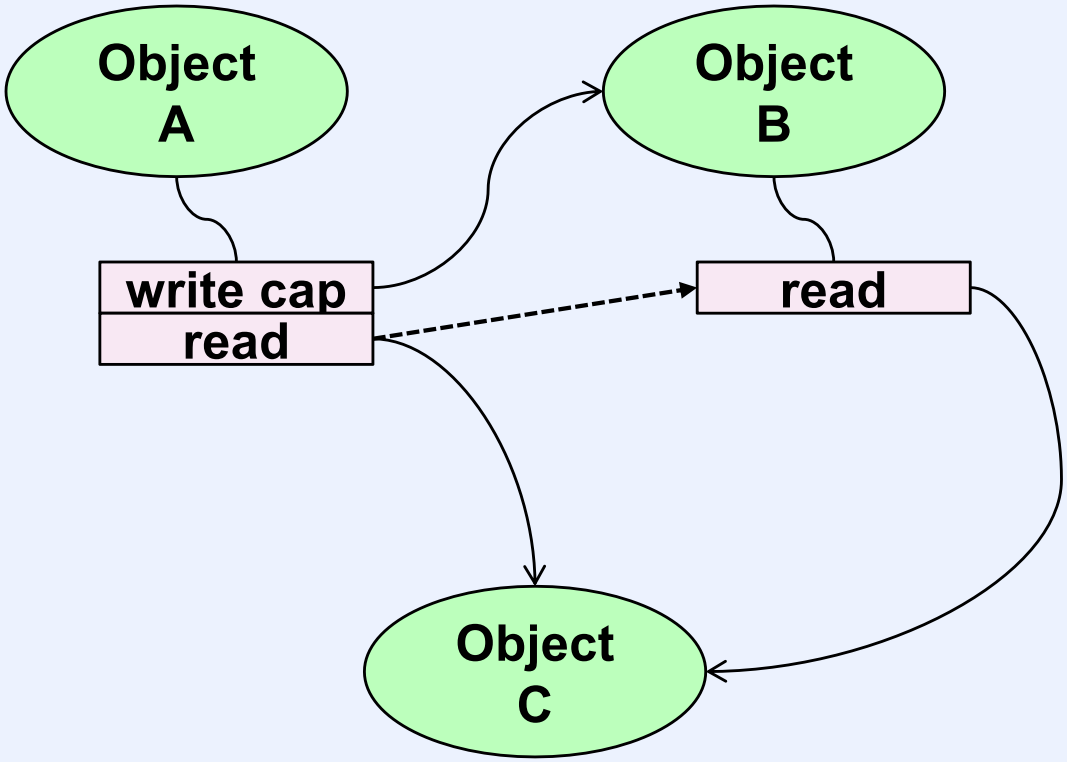
- **Subjects and resources are *objects* (in the OO sense)**



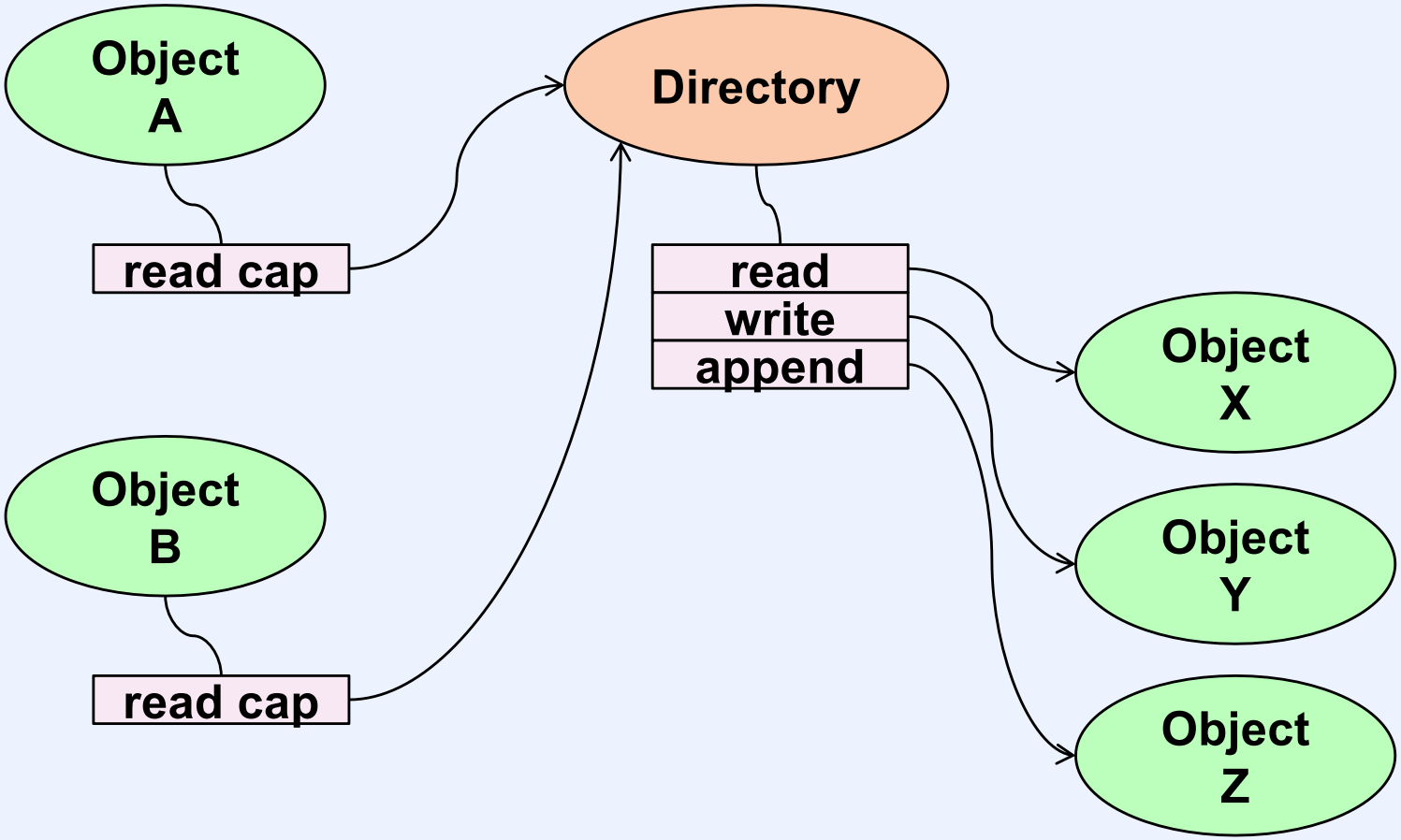
Copying Capabilities (1)



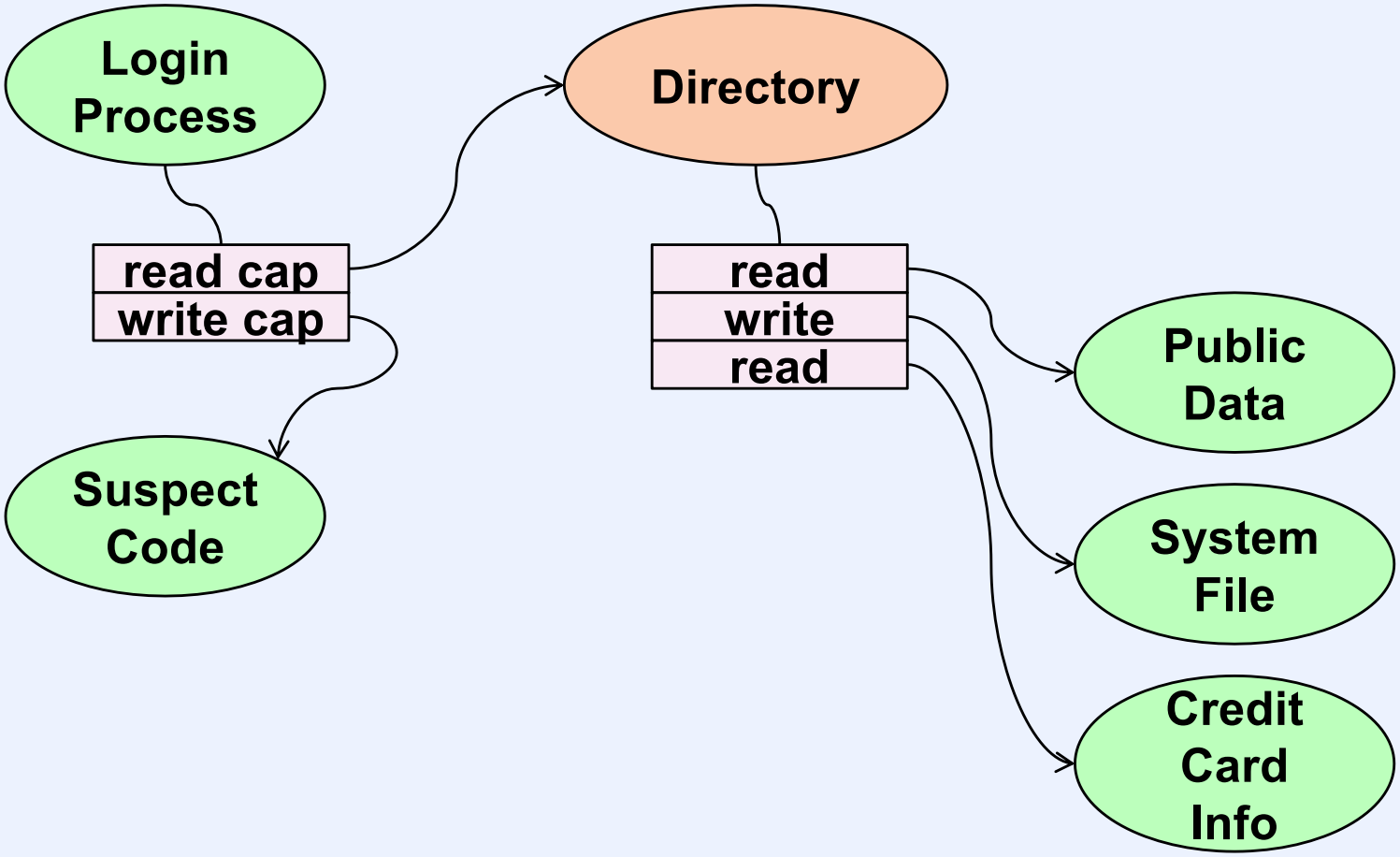
Copying Capabilities (2)



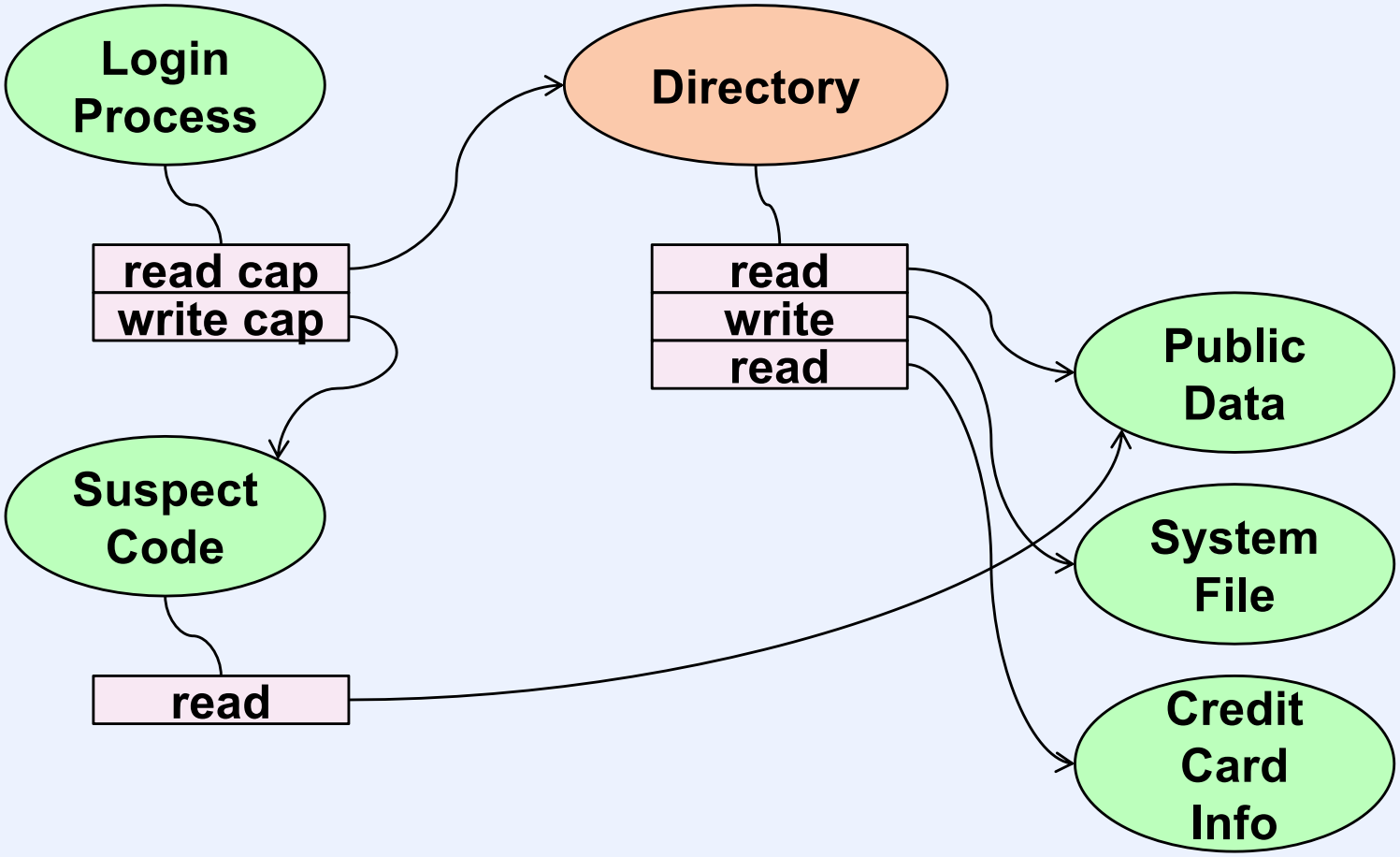
“Directories”



Least Privilege (1)



Least Privilege (2)

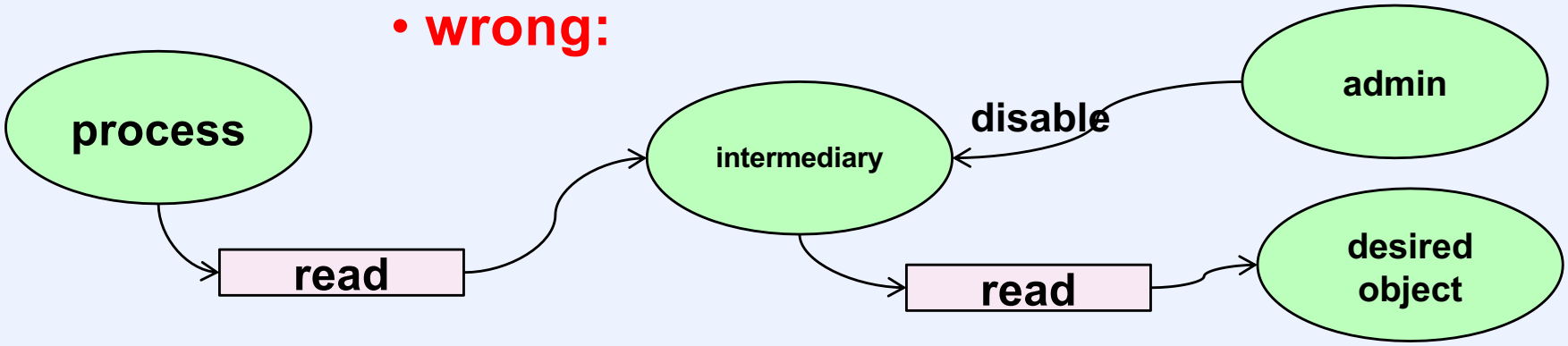


Issues

- Files aren't referenced by names. How do your processes get capabilities in the first place?
  - your "account" is your login process
    - created with all capabilities it needs
    - persistent: survives log-offs and crashes

Issues

- Can MAC be implemented on a pure capability system?
  - proven impossible twice
    - capabilities can be transferred to anyone
      - wrong: doesn't account for write-capability and read-capability capabilities
    - capabilities can't be retracted once granted
      - wrong:



Do Pure Capability Systems Exist?

- Yes!
  - long history
    - Cambridge CAP System
    - Plessey 250
    - IBM System/38 and AS/400
    - Intel iAPX 432
    - KeyKOS
    - EROS

A Real Capability System

- KeyKOS
  - commercial system
  - capability-based microkernel
  - used to implement Unix
    - (sort of defeating the purpose of a capability system …)
  - used to implement KeySafe
    - designed to satisfy “high B-level” orange-book requirements
    - probably would have worked
    - company folded before project finished

KeySafe