Security Part 5

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

Recap: TOCTTOU vulnerability

```
/* handin: a setuid-twd
   program */

if (access(argv[1],
    R_OK) == 0) {
  // ... fail
}
fd = open(argv[1],
  O_RDONLY);
/* copy argv[1] to course
   directory */
```

```
% handin my_assgn
…
```

Hidden Code

How to Solve?

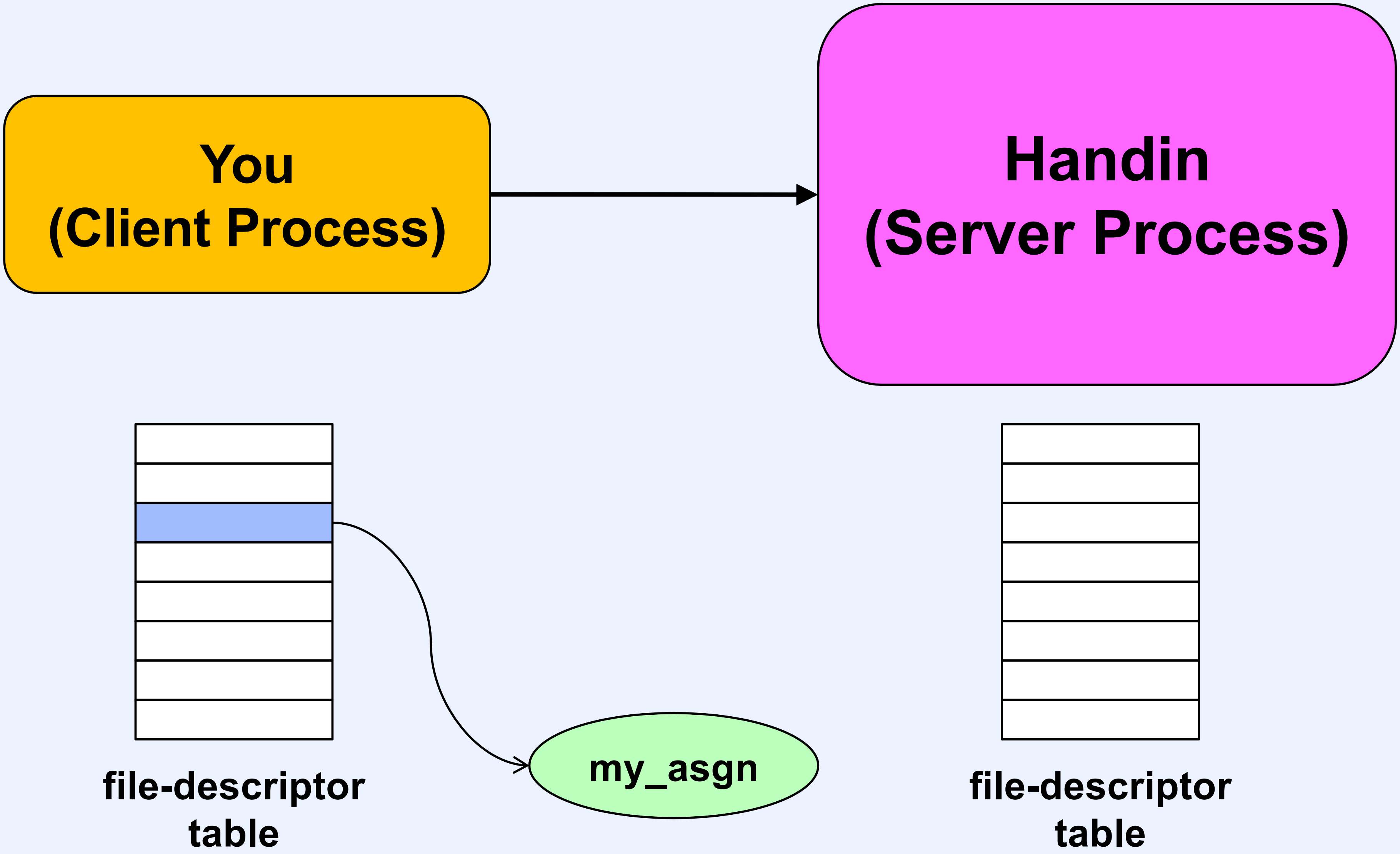
- **Could use previous solution**

  or

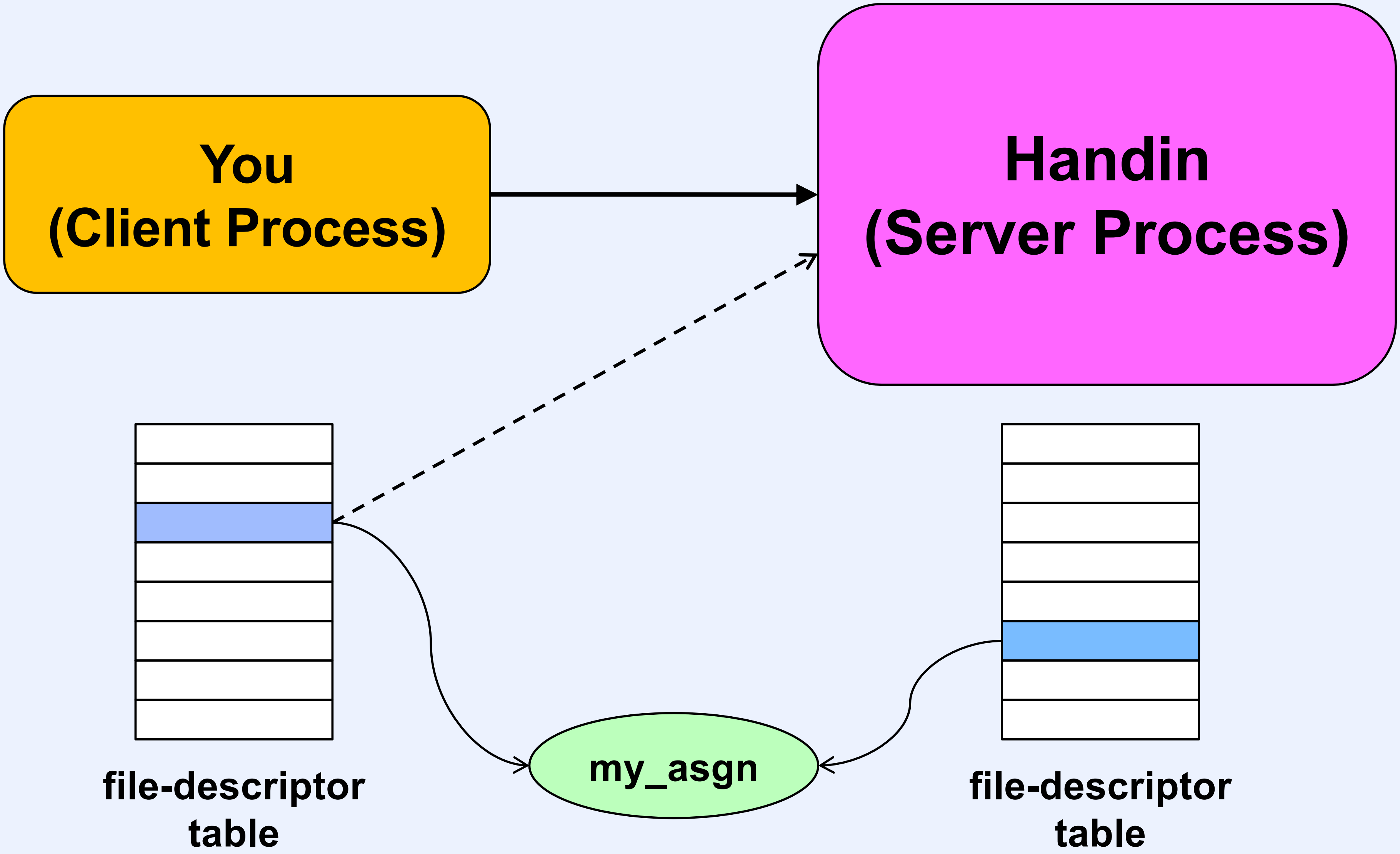
```
afd = open("my_assgn",
       O_RDONLY);
close(0);
dup(afd);
close(afd);
execl("handin", 0);
```

```
/* handin */
int main() {
   int user = getuid();
   char fname[256];
   sprint(fname,
         "CourseDirectory/%d", user);
   int ofd = open(fname,
         O_CREAT|O_WRONLY, 0666);
   while(1) {
      if ((c = read(0, buf, 256)) == 0)
         break;
      write(ofd, buf, c);
   }
   return 0
}
```

Same But Different



File Descriptor as *Capability*



Changing Security Context

Shell Commands

- su [user_name]
  - run a new shell with real and effective user IDs being those of user_name
    - if no user_name, then root (super user)
  - must supply correct password for user_name
- sudo program
  - run program with appropriate identity and privileges
  - checks to see if caller has permission
    - protected file lists who is allowed to do what
  - must supply your password

Programming Securely

- It's hard!
- Some examples …

Truncated Paths

```
int GetFile(char *dirpath, char *name) {
  char FullyQualifiedName[1024];
  if (CheckName(dirpath) == BAD) {
    ...
  }
  strncpy(FullyQualifiedName, dirpath, 512);
  strncat(FullyQualifiedName, name, 512);
  return(open(FullyQualifiedName, O_RDWR));
}

GetFile("//////////////////////…//tmp", vmlinuz);
```

Defense

- It’s not enough to avoid buffer overflow …
- Check for truncation!

Carelessness

```
char buf[100];
int len;
```

← Should be size_t

```
read(fd, &len, sizeof(len));
```

Read data size into len

```
if (len > 100) {
  fprintf(stderr, "bad length\n");
  exit(1);
}
```

Intention: check code doesn’t read too much

```
read(fd, buf, len);
```

Read actual data (len bytes)

A Real-Life Exploit …

- **sendmail -d6,50**
  - **means: set flag 6 to value 50**
  - **debug option, so why check for min and max?**
    - **(shouldn't have been turned on for production version …)**
    - **(but it was …)**
- **sendmail -d4294967269,117 -d4294967270,110 -d4294967271,113 changed *etc* to *tmp***
  - **/etc/sendmail.cf identifies file containing mailer program, which is executed as root**
  - **/tmp/sendmail.cf supplied by attacker**
    - **identifies /bin/sh as mailer program**
    - **attacker gets root shell**

What You Don't Know …

```
int TrustedServer(int argc, char *argv[]) {
   ...
   printf(argv[1]);
   ...
}
```

```
% TrustedServer "wxyz%n"
```

from the printf man page:

%n The number of characters written so far is stored into the integer indicated by the int * (or variant) pointer argument. No argument is converted.

→ `printf` changes arbitrary (?) memory location

Does This Work?

```
% setenv LD_PRELOAD myversions/libcrypt.so.1
% su
Password:
```

Isolating Security Contexts

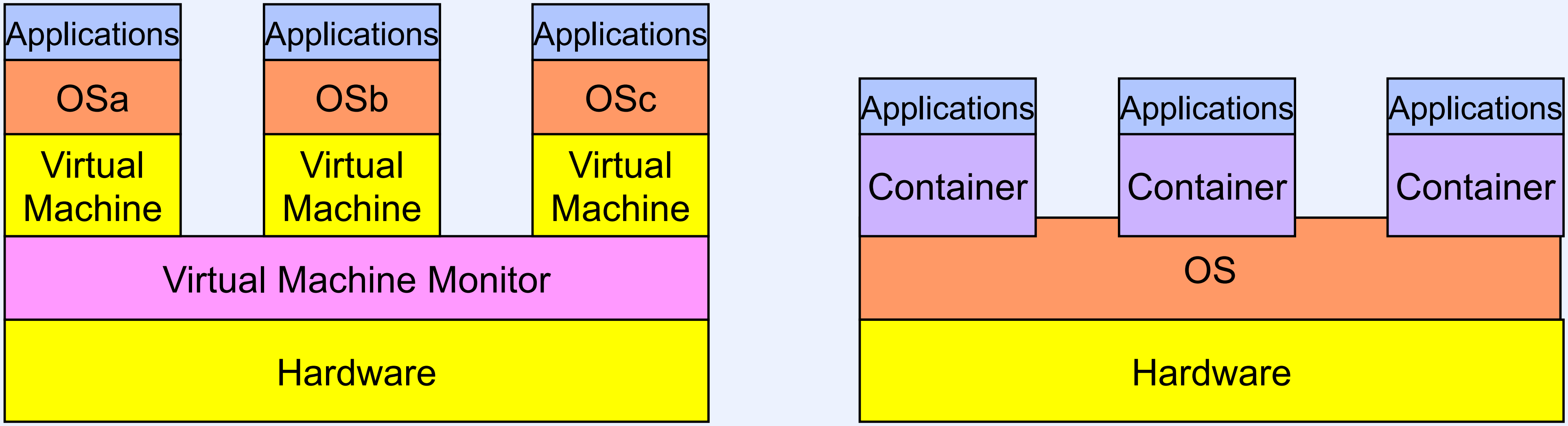
Principle of Least Privilege

- Perhaps:
  - run process with a minimal security context
    - special account, etc.
  - send it the capabilities it needs

Complete Isolation

- **Would like to run multiple applications in complete isolation from one another**
  - **run them on separate computers with no common file system**
  - **run them on separate virtual machines**
  - **run them in separate *containers* on one OS instance**

VMs versus Containers



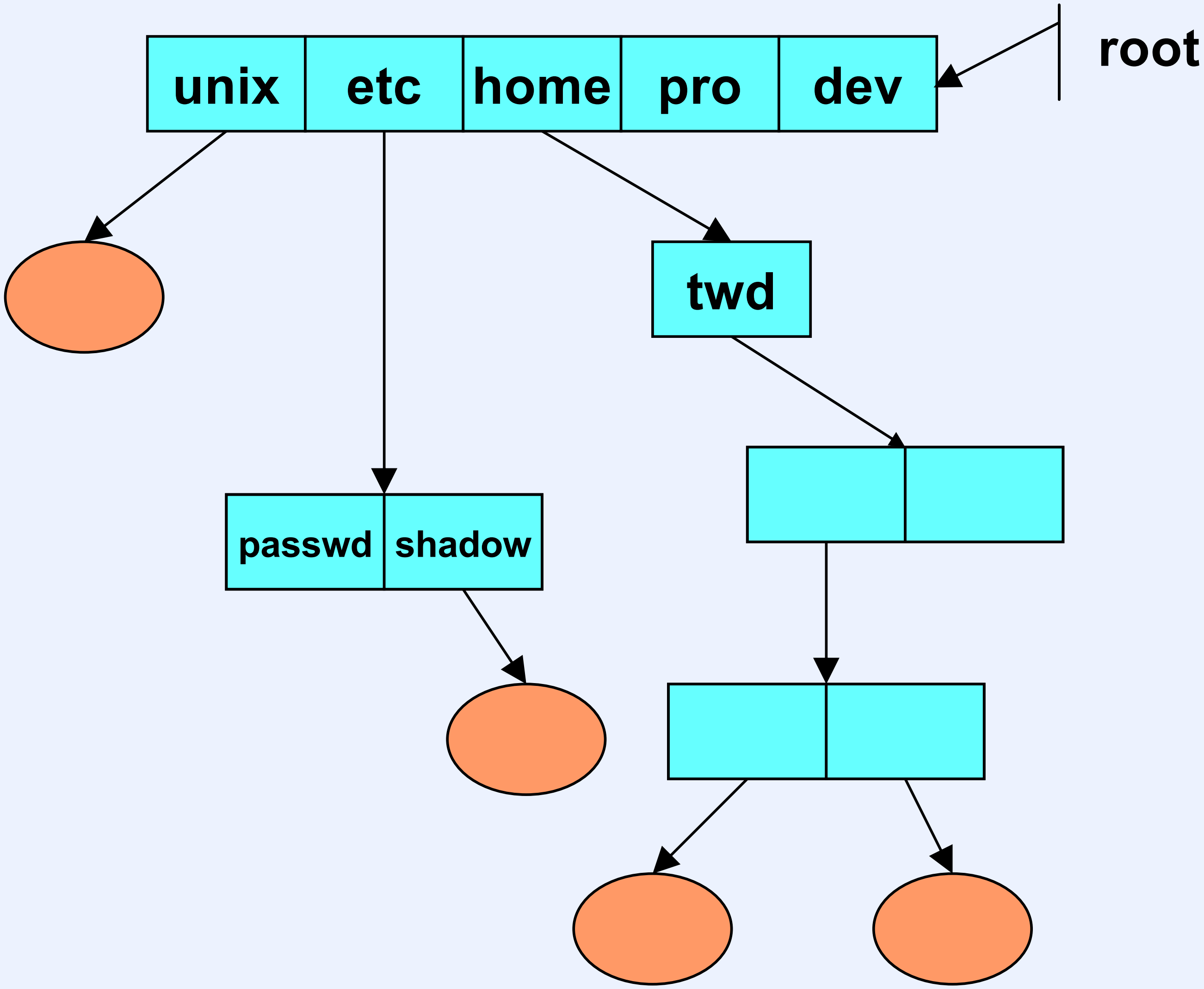
Containers

- Isolated
  - processes in a container can't access what's not in the container
  - processes in a container shouldn't even be aware of what's not in the container

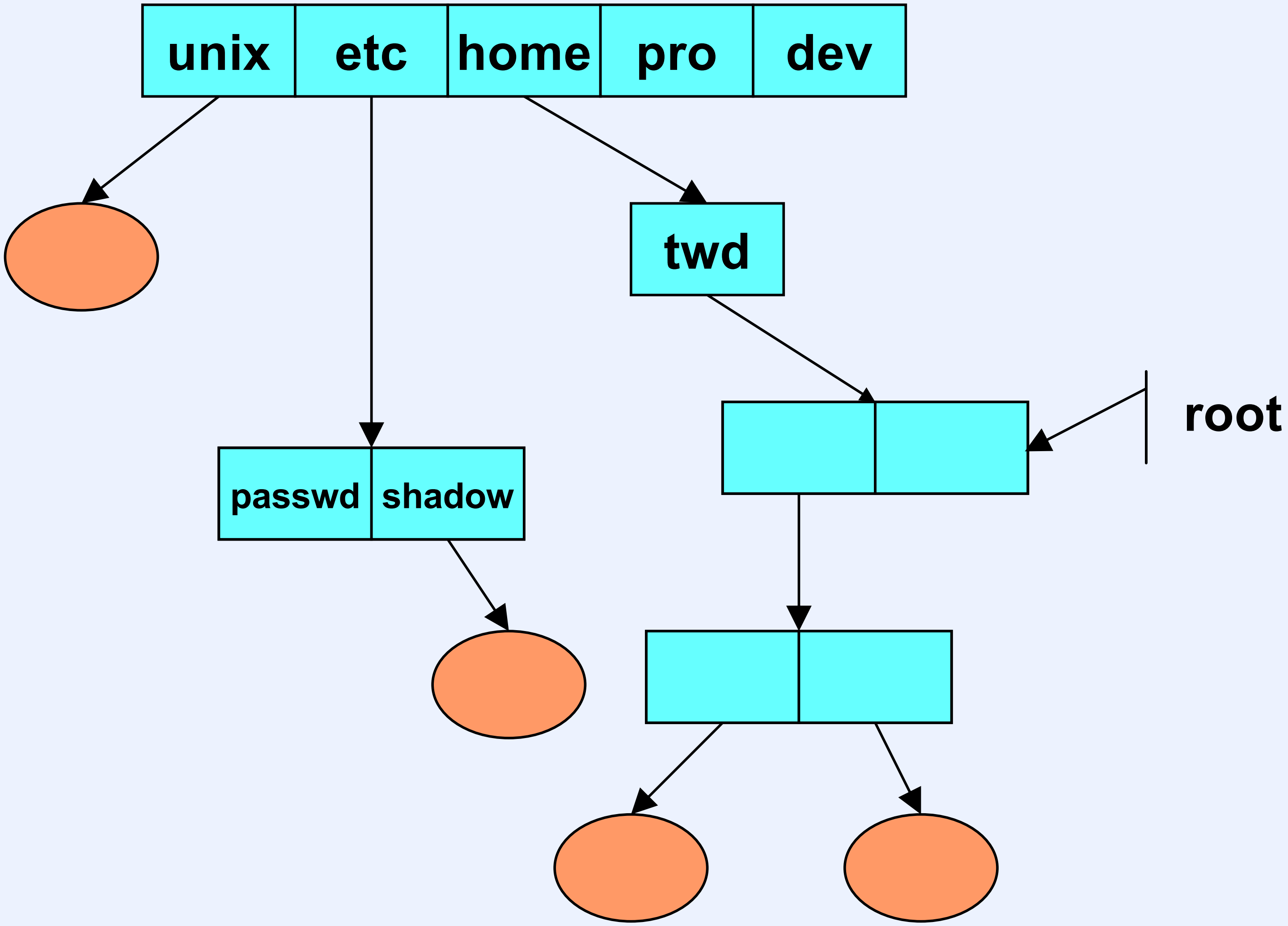
Container Building Blocks

Part 1: File system (chroot)

chroot (before)



chroot (after)



Not a Quiz

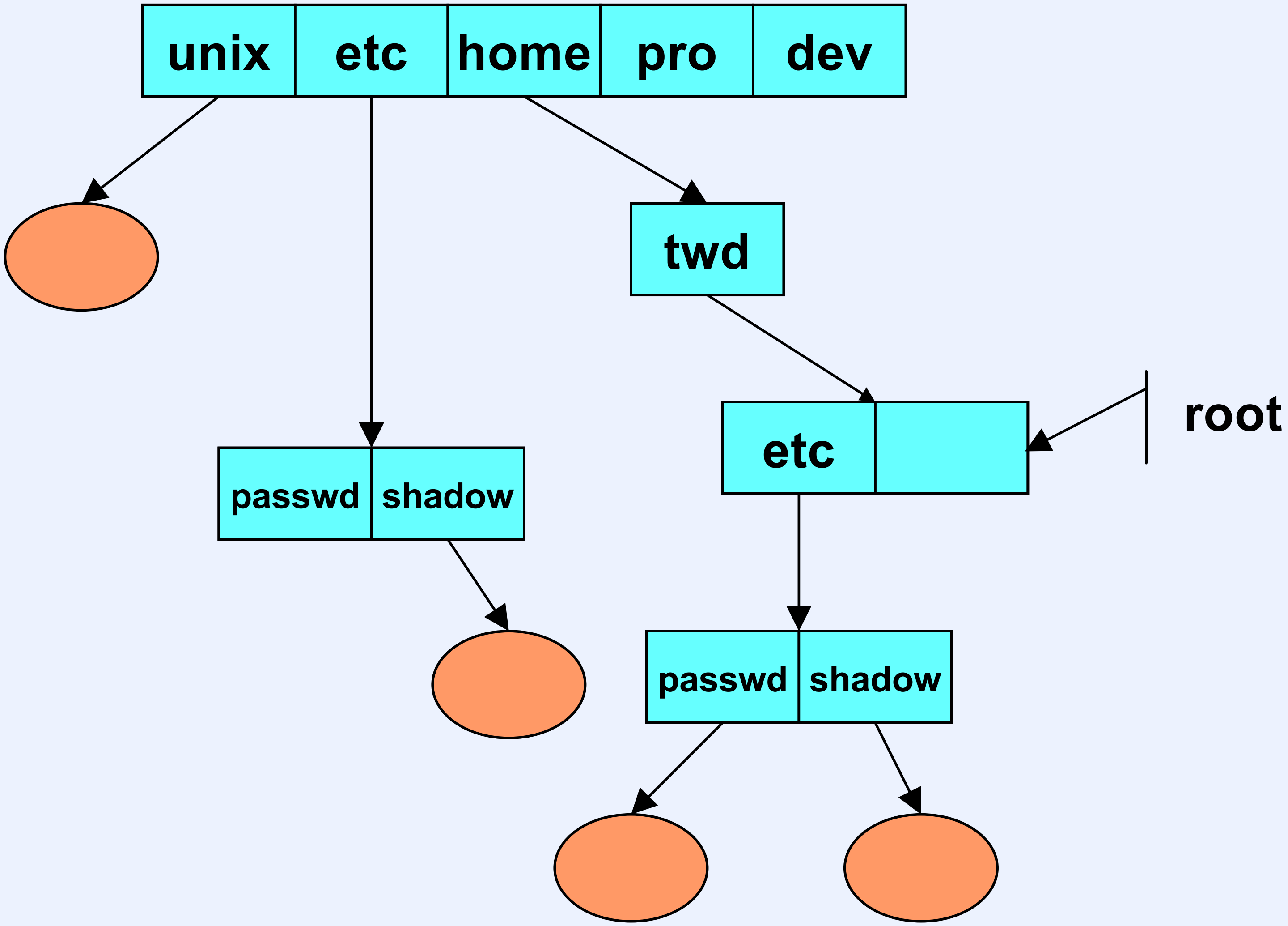
Restricting a process to a particular subtree

a) improves security by effectively running the process in a smaller protection domain

b) has little effect on security

c) potentially makes security worse

chroot (after)



Relevant System Calls

- `chroot(path_name)`

- `chdir(path_name)`
- `fchdir(file_descriptor)`

Not a Quiz

After executing *chroot*, “/” refers to the process’s new root directory. Thus “..” is the same as “.” at the process’s root, and the process cannot cd directly to the “parent” of its root. Also, recall that hard links may not refer to directories.

a) *chroot* does effectively limit a process to a subtree

b) *chroot* does not effectively limit a process to a subtree

Escape!

```
chdir("/");
pfd = open(".", O_RDONLY);
mkdir("Houdini", 0700);
chroot("Houdini");
fchdir(pfd);
for (i=0; i<100; i++)
  chdir("..");
chroot(".");
```

Namespace Isolation

- Isolate process by restricting it to a subtree
  - chroot isn’t foolproof
- Fix chroot
  - make it superuser only
  - make sure processes don’t have file descriptors referring to directories above their roots

Fixed in BSD

- jail
  - can't *cd* above root
  - all necessary files for standard environment present below root
  - *ps* doesn't see processes in other jails

Container Building Blocks

Part 2: Resources & Namespaces

Linux Responds ...

- cgroups
  - – group together processes for
    - resource limiting
    - prioritization
    - accounting
    - control
- name-space isolation
  - – isolate processes in different name spaces
    - mount points
    - PIDs
    - UIDs
    - etc.

Linux Containers

- Reside in isolated subtrees
  - (fixed) chroot restricts processes in a container to the subtree
  - file systems are mounted in container namespaces, so that other containers can't see them
- Separate UID and PID spaces
  - PIDs start at 1 for each container
  - container UIDs mapped to OS UIDs
    - UID 0 has privileges in container, but not outside of container
- Limits placed on CPU, I/O and other usages

Docker

- Runs in Linux containers (also runs on Windows)
  - container contains all software and files needed for execution
  - provides standard API for applications
    - even if on Windows
- On macOS, actually runs a hypervisor and runs containers inside (Linux) VM

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 (starting with Vista)
  - accounts with administrator privileges have two access tokens
    - one for normal usage
    - another with elevated rights

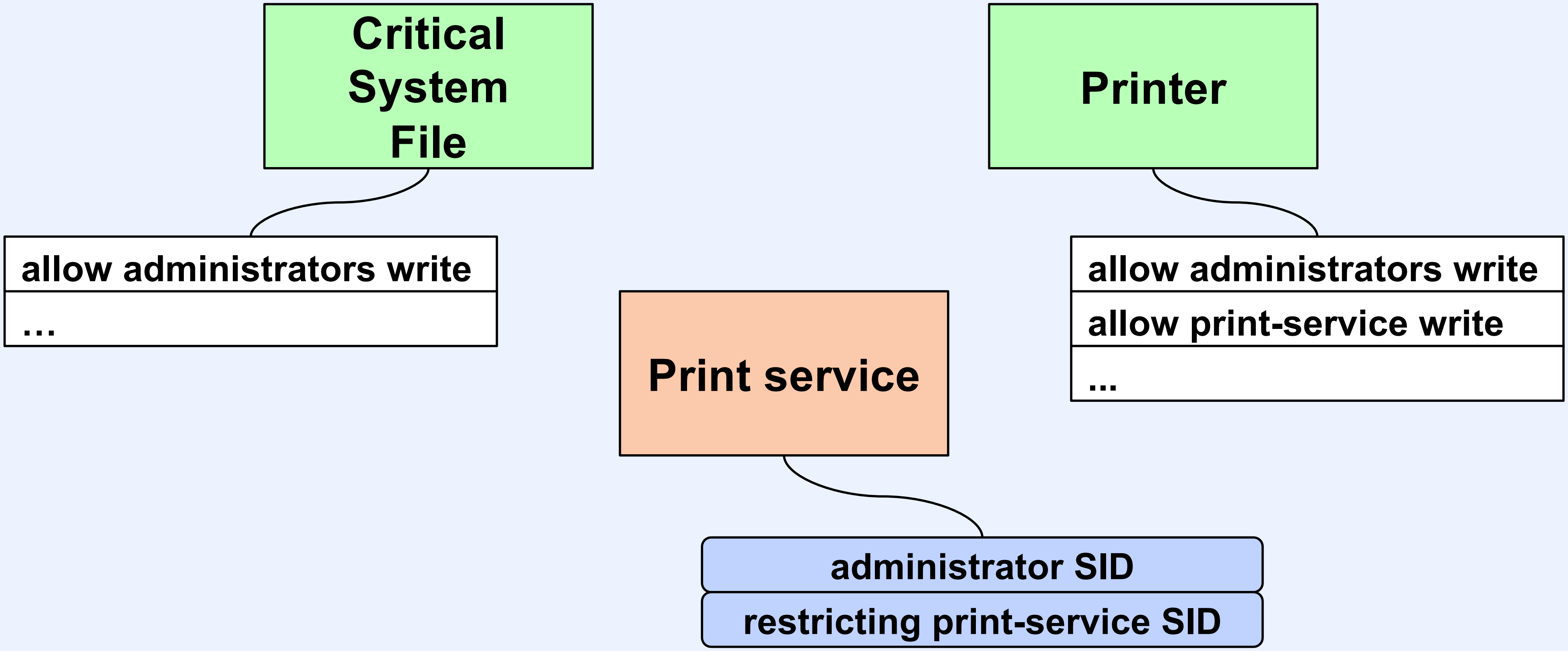
Least Privilege

- 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



Critical System File
allow administrators write
…
Printer
allow administrators write
allow print-service write
...
Print service
administrator SID
restricting print-service SID

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

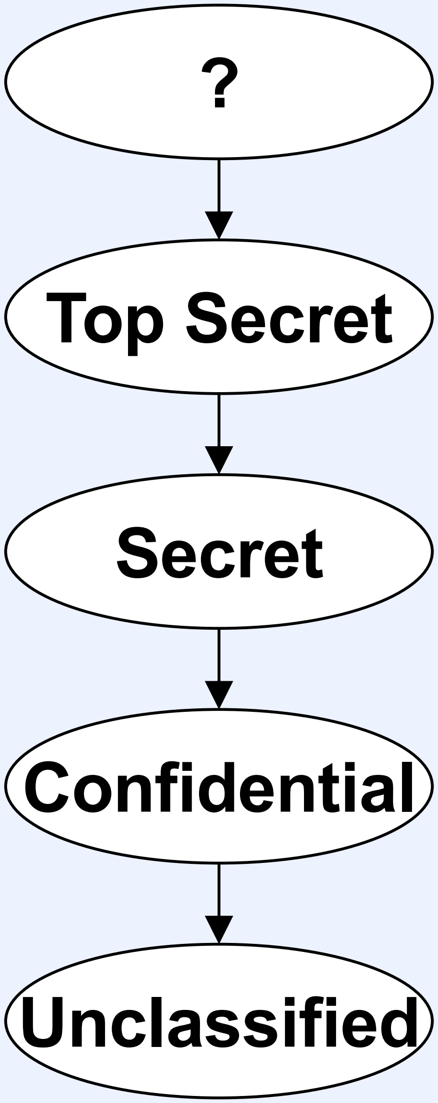
Serious Security

- National defense
- Proprietary information
- Personal privacy

Mandatory vs. Discretionary Access Control

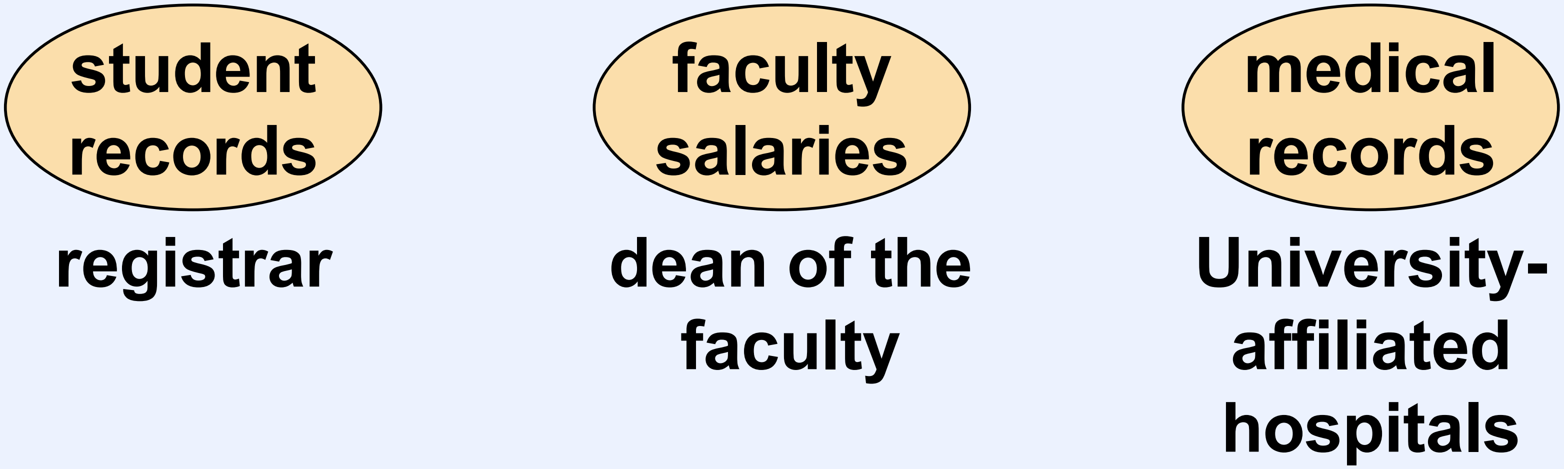
- Discretionary
  - ACLs, capabilities, etc.
    - access is at the discretion of the owner
- Mandatory
  - government/corporate security, etc.
    - access is governed by strict policies

Mandatory Access Control (1)



Mandatory Access Control (2)

- Privacy/confidentiality policies
  - compartmentalization



student records

registrar

faculty salaries

dean of the faculty

medical records

University-affiliated hospitals

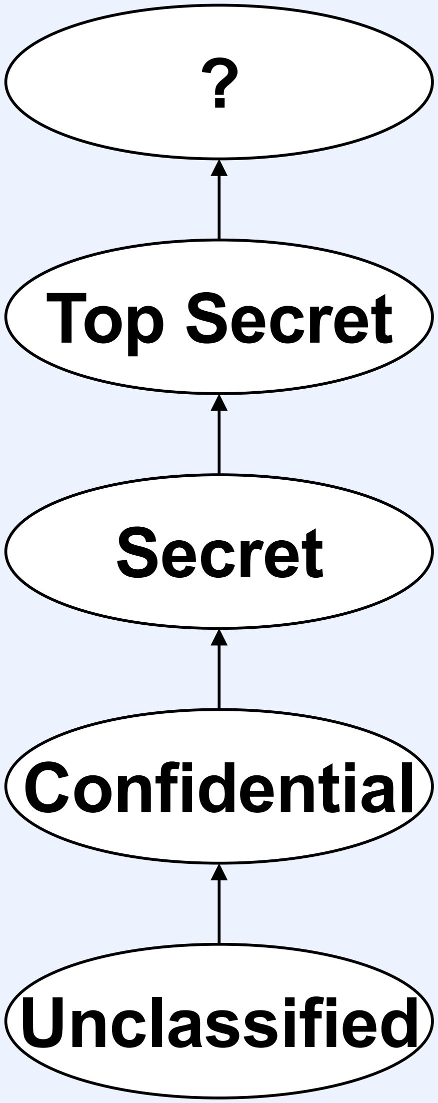
Mandatory Access Control (3)

- 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



?

Top Secret

Secret

Confidential

Unclassified

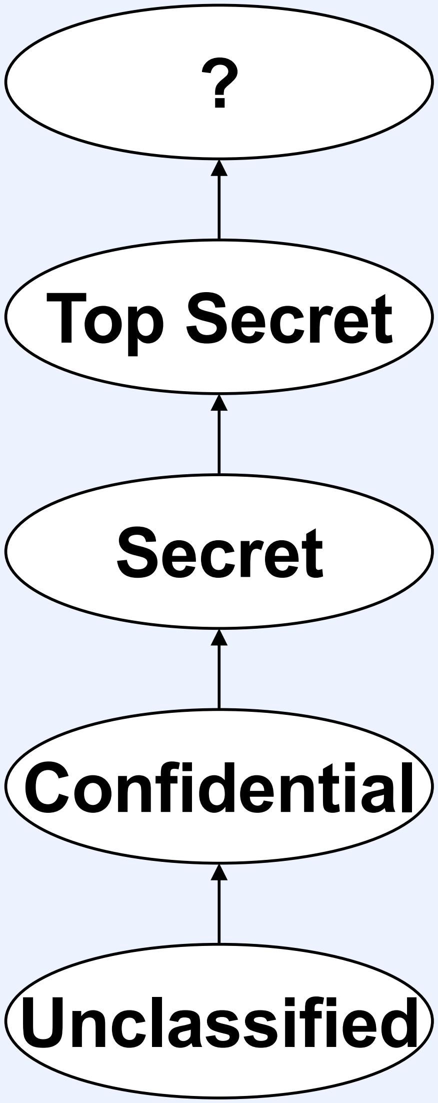
Attack!

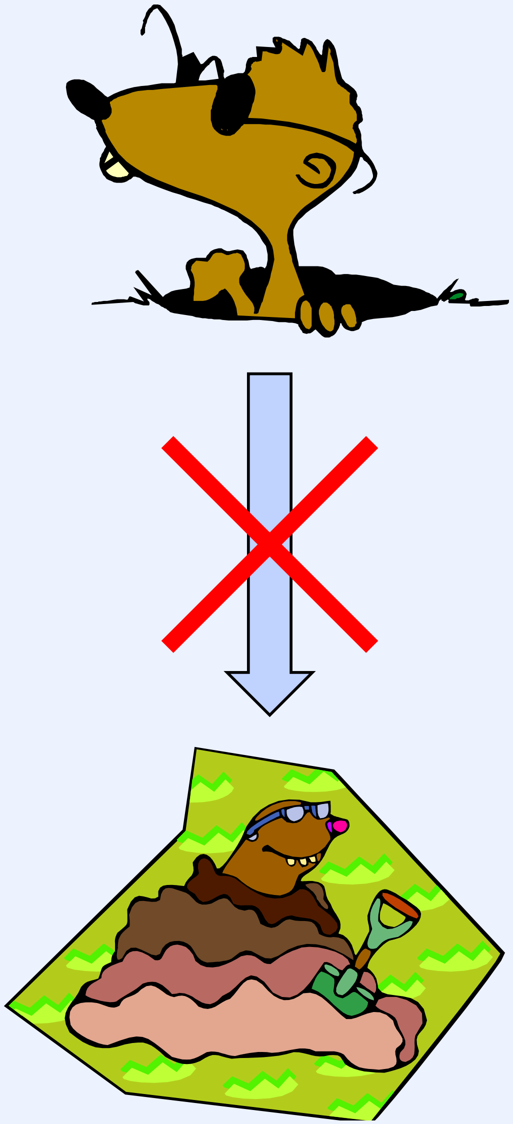
Not cleared for top-secret orders

Managing Confidentiality

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

Espionage



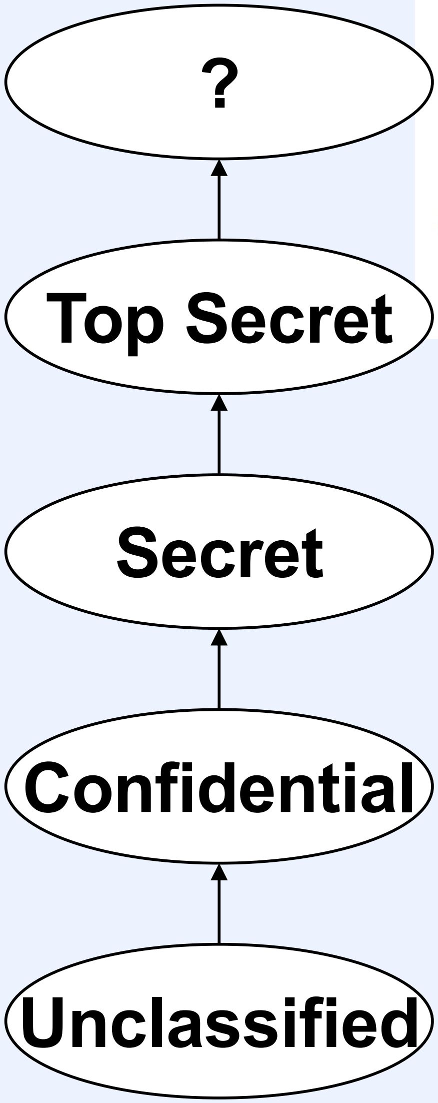


agent X learns of invasion plans

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

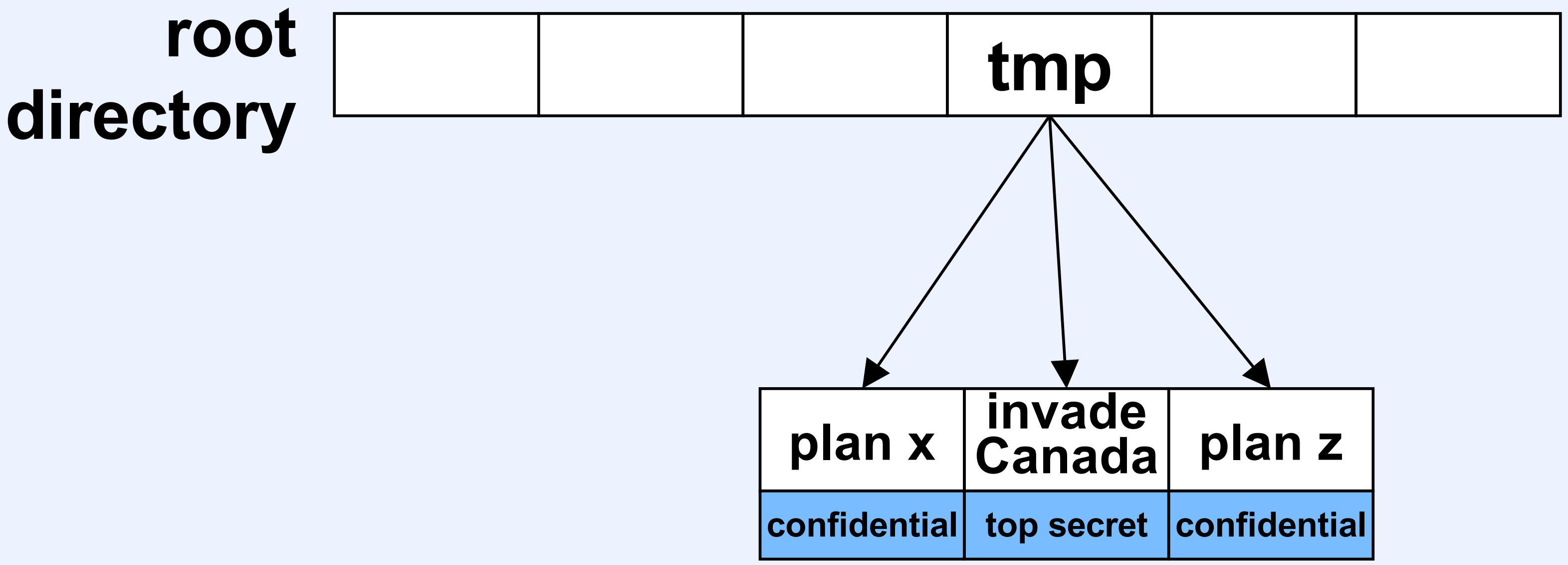
sneaky communication possible

agent Y monitors load sends email to spymaster

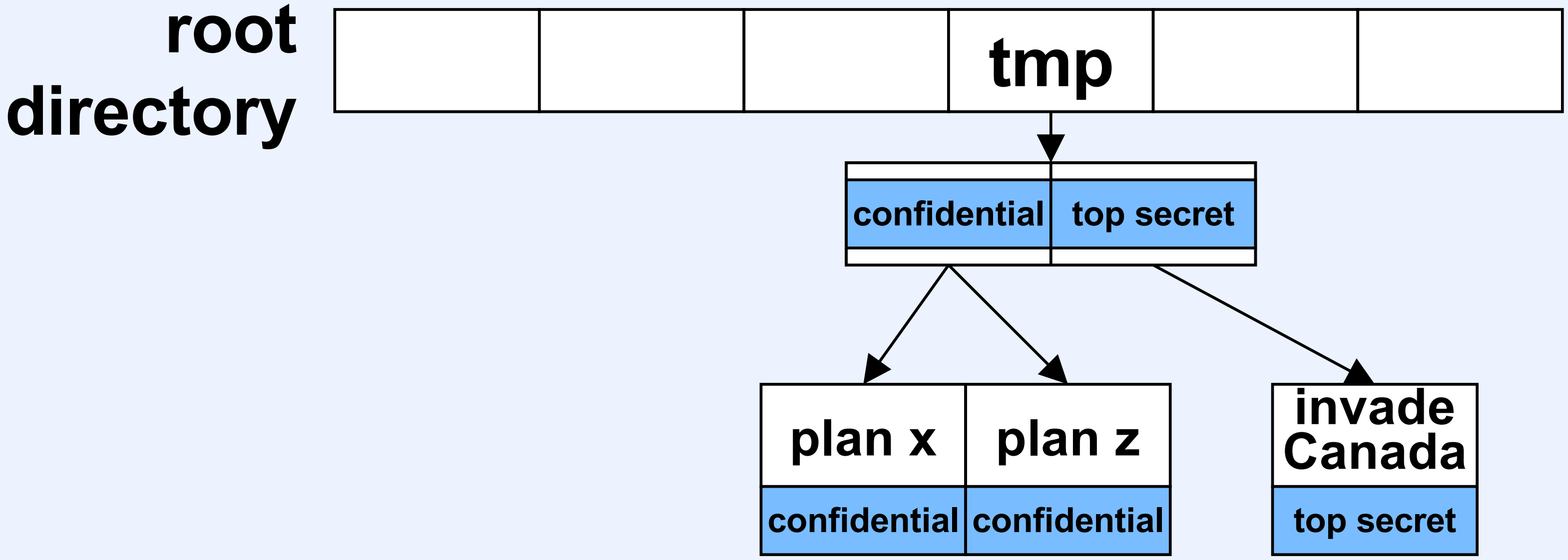
Defense

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

Multi-Level Directories (1)



Multi-Level Directories (2)


root
directory
tmp
confidential
top secret
plan x
plan z
invade Canada
confidential
confidential
top secret