Security Part 4

Unix ACLs

- **Access checking**
  - **if effective user ID of process matches file's owner**
    - ***user_obj entry* determines access**
  - **if effective user ID matches any *user ACE***
    - ***user entry* ANDed with *mask* determines access**
  - **if effective group ID or supplemental group matches file's group or any *group ACE***
    - **access is intersection of *mask* and the union of all matching group entries**
  - **otherwise, *other ACE* determines access**

Example

```
% mkdir dir
% ls -ld dir
drwxr-x--- 2 twd fac 8192 Mar 30 12:11 dir
% setfacl -m u:floria:rwx dir
% ls -ld dir
drwxr-x---+ 2 twd fac 8192 Mar 30 12:16 dir
% getfacl dir
# file: dir
# owner: twd
# group: cs-fac
user::rwx
user:floria:rwx
group::r-x
mask::rwx
other::---
```

Quiz 1

Unlike Windows ACLs, UNIX ACLs have no deny entries. Is it possible to set up an ACL that specifies that everyone in a particular group has rw access, except that a certain group member has no access at all?

a) No, it can't be done

b) Yes, it can be done in two commands

c) Yes, but it's complicated and requires more than two commands

Real-world Problem: Cross-OS ACL Interoperability

NFSv4 ACLs

- NFSv4 designers wanted ACLs
  - on the one hand, NFS is used by Unix systems
  - on the other hand, they'd like it to be used on Windows systems
  - solution:
    - adapt Windows ACLs for Unix
    - NFSv4 servers handle both Unix and Windows clients
    - essentially Windows ACLs plus Unix notions of file owner and file group

ACLs at Brown CS (Up Till Fall 2019)

- Linux systems support POSIX ACLs
- Windows systems support Windows ACLs
- Servers run GPFS file system and handle NFSv3 and SMB clients
  - GPFS supports NFSv4 ACLs
  - translated to POSIX ACLs and Unix bit vectors for NFSv3 clients
  - translated to Windows ACLs for SMB clients

ACLs at Brown CS (What was Planned for Fall 2019)

- Switch to Isilon servers managed by CIS
  - support NFSv4 and SMB clients
- Linux and Mac clients use NFSv4
  - switch to NFSv4 ACLs
- Windows clients use SMB

OSX (Macs)

- Native support for NFSv4 ACLs
  - no setfacl/getfacl commands, but built into chmod
- No NFSv4 client support
  - third-party implementations exist, but they don’t work

ACLs at Brown CS (Fall 2019 – Present)

- Isilon servers managed by OIT
  - support NFSv4 and SMB clients
- Linux clients use NFSv4
  - switch to NFSv4 ACLs
- Windows clients use SMB
- OSX clients use SMB
  - no groups – just the authenticated user
  - all remote files seen as allowing 0700 access
  - clients can't observe or modify access protection
    - (though still enforced on server)

Advanced Access Control

setuid and friends

Extending the Basic Models

- Provide a file that others may write to, but only if using code provided by owner
- Print server
  - pass it file names
  - print server may access print files if and only if client may
- Password-changing program (`passwd`)

Superuser (Unix)

- User ID == 0
  - bypasses all access checks
  - can send signals to any process

Attaining Super (or Lesser) Powers

- Setuid protection bit
  - the exec'ing process's UID is set to owner of file

User and Group IDs

- *Real* user and group IDs — usually used to identify who created the process
- *Effective* user and group IDs — used to determine access rights to files
- Saved user and group IDs — hold the initial effective user and group IDs established at the time of the *exec*, allowing one to revert back to them

Exec

- Normally the real and effective IDs are the same
  - they are copied to the child from the parent during a *fork*
- *exec*s done on files marked *setuid* or *setgid* change this
  - if the file is marked *setuid*, then the effective and saved user IDs become the ID of the owner of the file
  - if the file is marked *setgid*, then the effective and saved group IDs become the ID of the group of the file

Exercise of Powers

- Permission to access a file depends on a process's effective IDs
  - the *access* system call checks permissions with respect to a process's real IDs
    - this allows *setuid*/*setgid* programs to determine the privileges of their invokers
- The *kill* system call makes use of both forms of user ID; for process *A* to send a signal to process *B*, one of the following must be true:
  - *A*'s real user ID is the same as *B*'s real or saved user ID
  - *A*'s effective user ID is the same as *B*'s real or saved user ID
  - *A*'s effective user ID is 0

Race Conditions

```
// a setuid-root program:

if (access("/tmp/mytemp",
    W_OK) == 0) {
  // ... fail
}
fd = open("/tmp/mytemp",
    O_WRITE|O_APPEND);
len = read(0, buf,
    sizeof(buf));
write(fd, buf, len);
```

```
// another program:

unlink("/tmp/mytemp");
symlink("/etc/passwd",
    "/tmp/mytemp");
```

- **TOCTTOU vulnerability**
  - **time of check to time of use …**

Changing Identity (1)

- The *setuid* and *setgid* system calls give a process a limited ability to change its IDs

```
int setuid(uid_t uid)

int setgid(gid_t gid)
```

  – if the caller is super user, then these calls set the real, effective, and saved IDs
  – otherwise, these calls set only the effective IDs and do so only if the caller's real, saved, or effective ID is equal to the argument

Changing Identity (2)

- The *seteuid* and *setegid* system calls are the same except that they change only the effective IDs
- The system calls *getuid*, *getgid*, *geteuid*, and *getegid* respectively return the real user ID, the real group ID, the effective user ID, and the effective group ID of the caller

Avoiding the Race Condition

```
uid_t caller_id = getuid();
uid_t my_id = geteuid();
seteuid(caller_id);
fd = open("/tmp/mytemp", O_WRITE|O_APPEND);
if (fd == -1) {
  // fail …
}
seteuid(my_id);
len = read(0, buf, sizeof(buf));
write(fd, buf, len);
```

Unix Security Context

- Security context of a process
  - real user and group IDs
  - effective user and group IDs
  - saved user and group IDs
  - more?

More …

- – supplementary groups
- – alternate root
- – file-descriptor table
- – privileges
  - - *super user* at finer granularity
  - - called capabilities in Linux

Quiz 2

```
/* 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
```

This adds my_assign to the course directory.

a) It works every time
b) It might fail occasionally (nothing gets added)
c) It might fail occasionally (something else gets added)
d) It never works

```
/* 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?

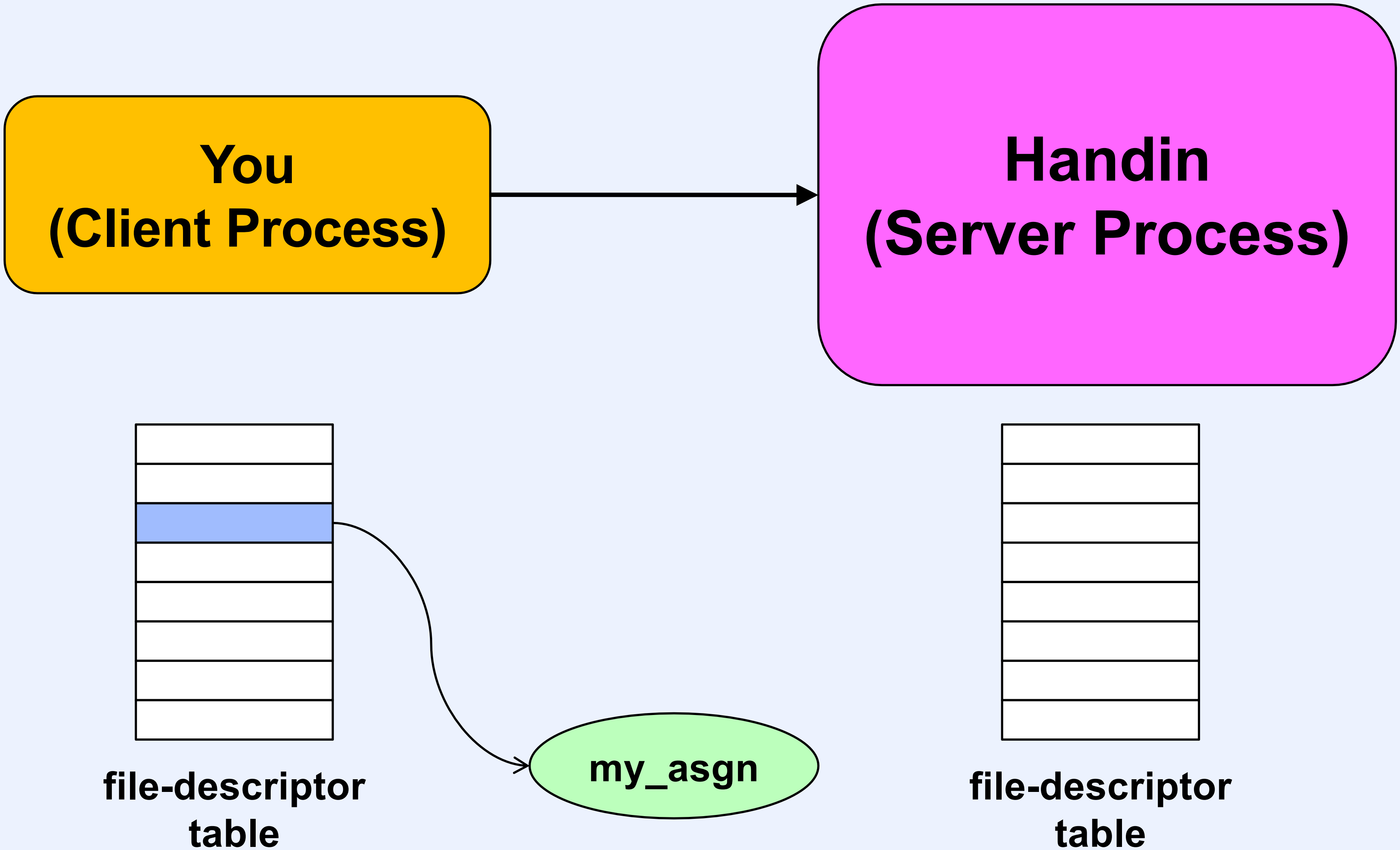
- **Could use previous solution**

  or

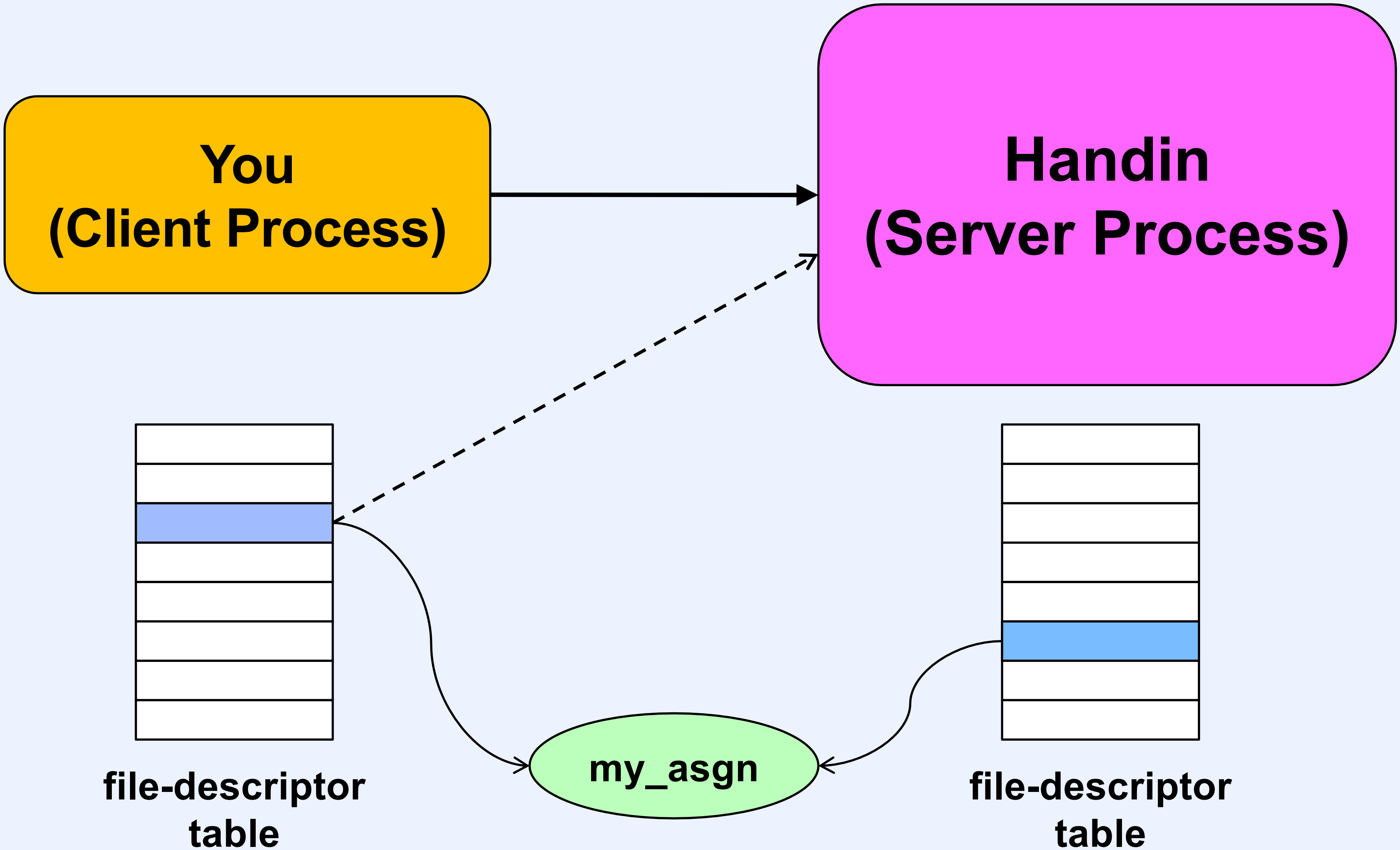
```
afd = open("my_assgn",
       O_RDONLY);
close(0);
dup(afd);
close(afd);
execl("handin", 0);
```

```
/* 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;

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

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

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

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.

Does This Work?

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

Isolating Security Contexts

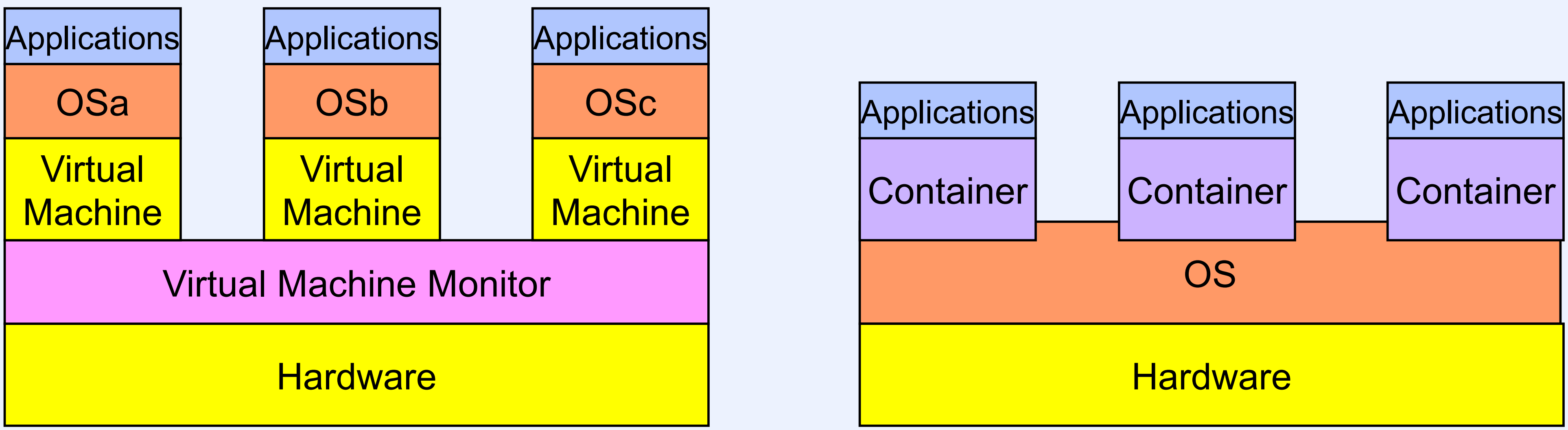
Principle of Least Privilege

- 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



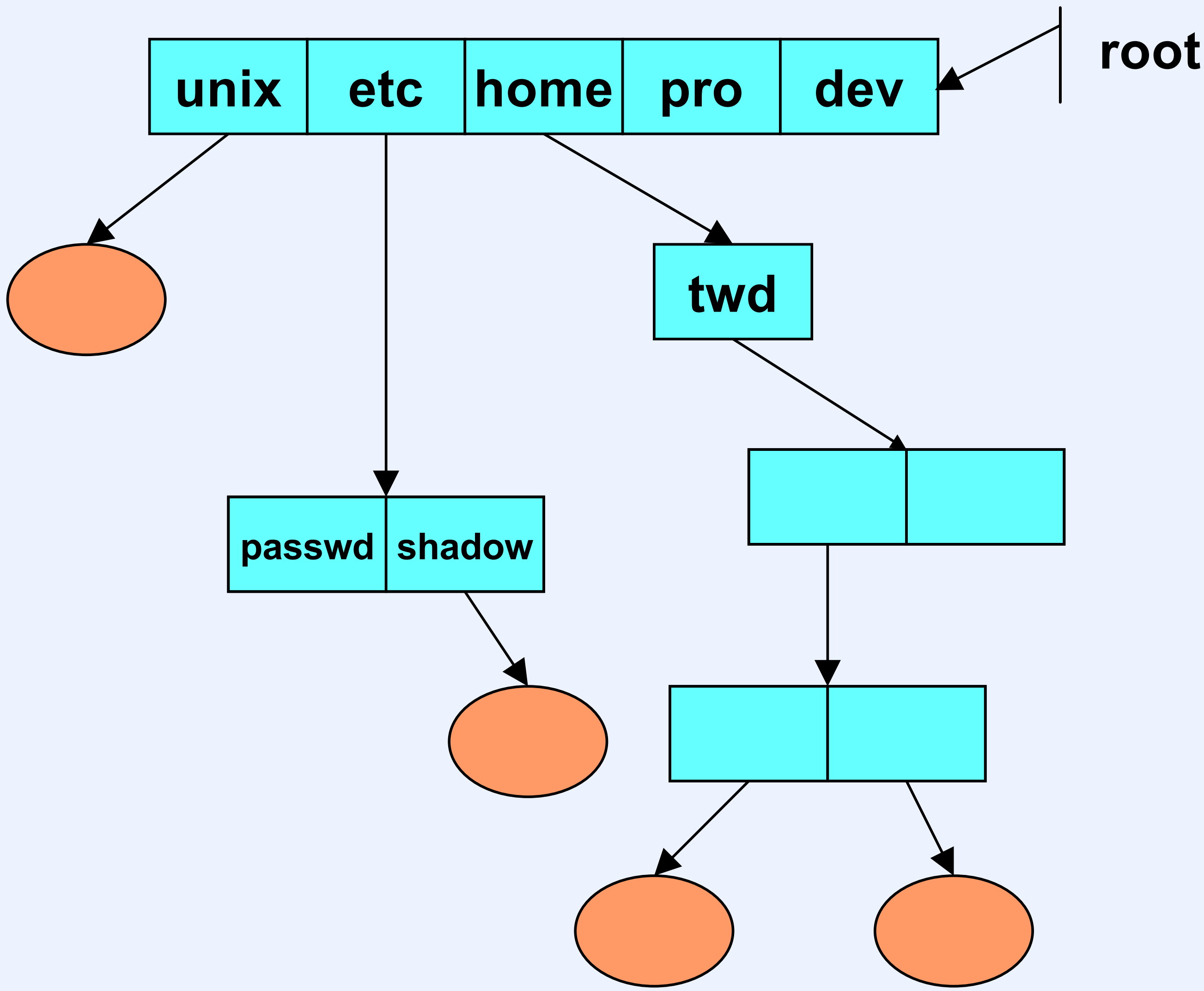
Containers

- 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

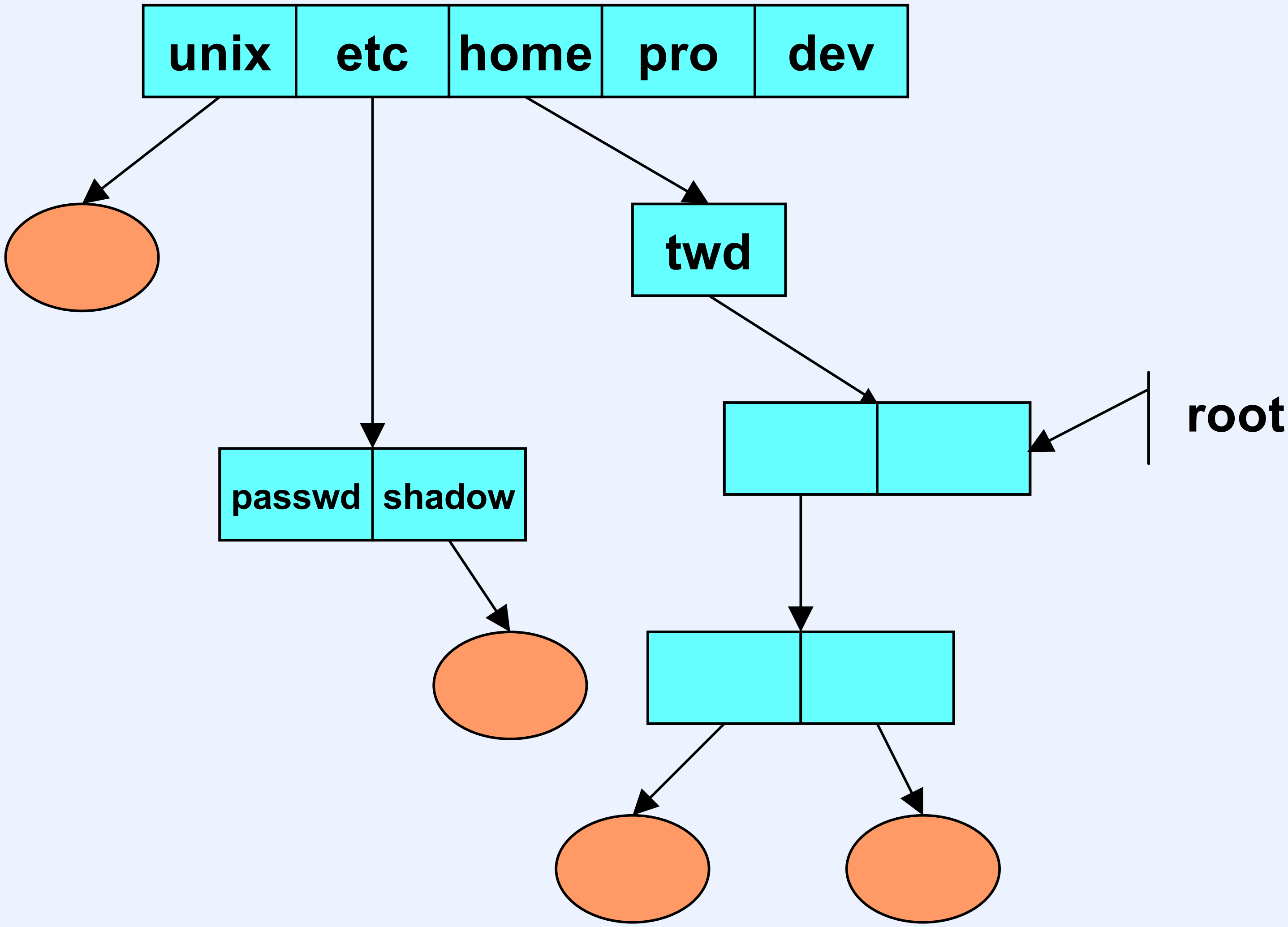
Container Building Blocks

Part 1: File system (chroot)

chroot (before)



chroot (after)

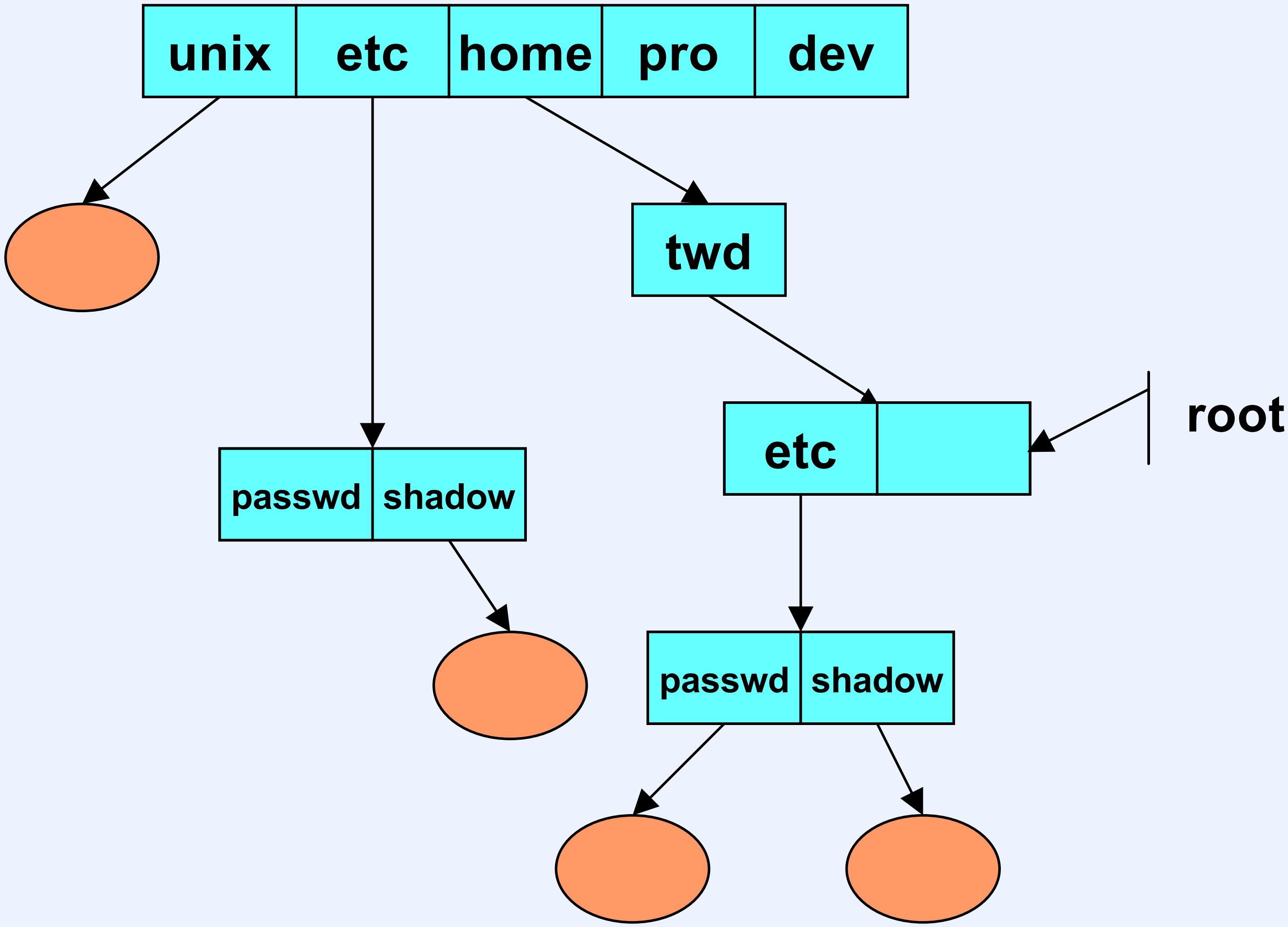


Not a Quiz

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