Access Control and Processes

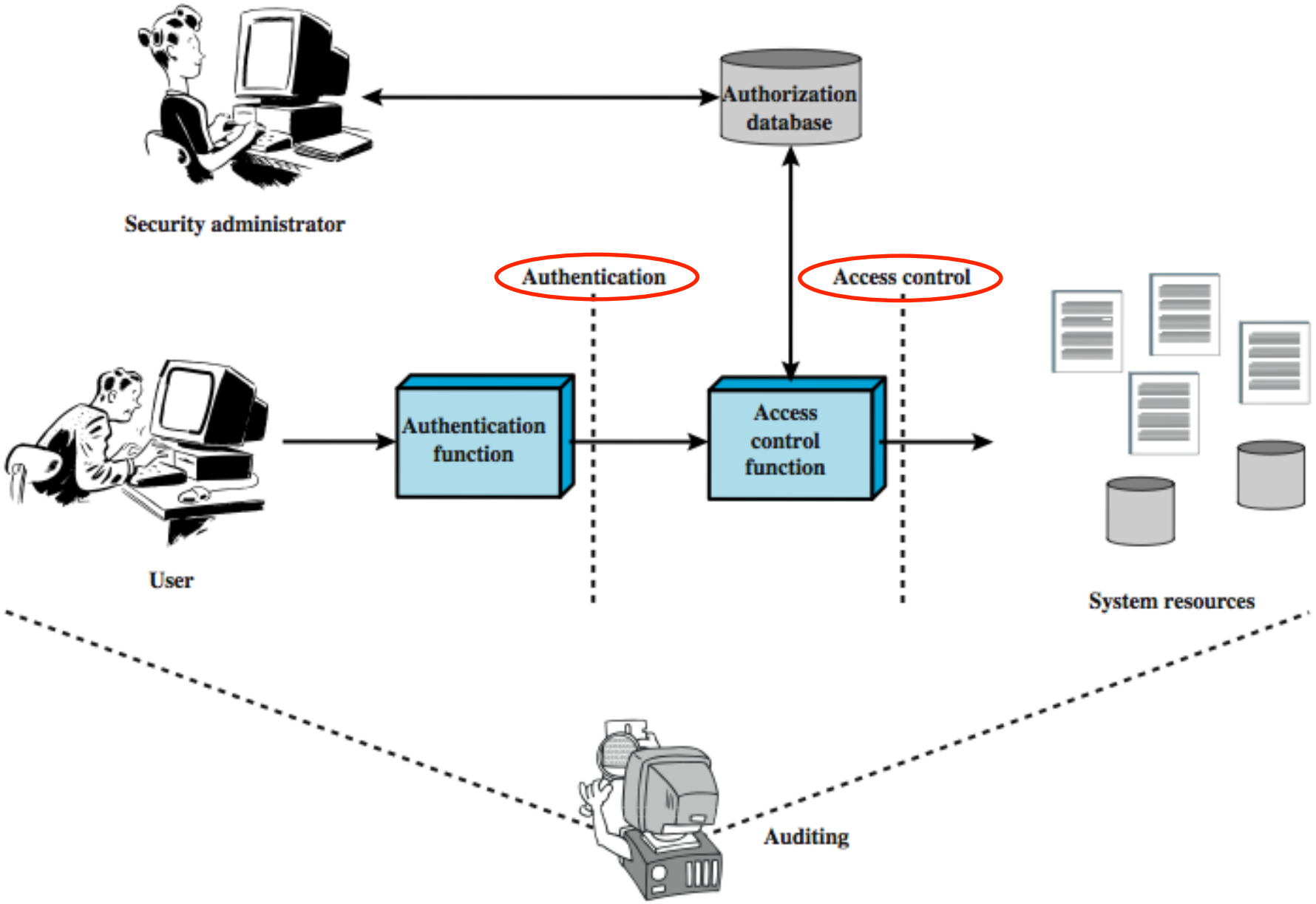
Portions courtesy Ellen Liu

Outline

- Access control
- Traditional UNIX access control
  - File system access control; File permissions, Some commands; The root account
  - Modern access control
- Controlling processes (1)
  - Components of a process; life cycle of a process

Access Control

- "The prevention of unauthorized use of a resource, including the prevention of a use in an unauthorized manner"
- Often decomposed into:
  - Authentication: Who are you?
  - Authorization: Can you take action X on resource Y?


Security administrator
Authorization database
Authentication
Access control
Authentication function
Access control function
User
System resources
Auditing

Access control elements

- **Subject** - entity that can access objects
  - Mainly a process representing user/application
  - 3 common classes of subject in basic access control systems: owner, group, world
- **Object** - access controlled resource
  - e.g. files, directories, records, programs, etc.
  - number/type depend on environment
- **Access right** - way in which a subject accesses an object
  - e.g., read, write, execute, delete, create, search

Three classes of objects

Owner: may be the creator of a resource such as a file. For system resources, ownership may belong to a system administrator. For project resources, a project administrator or leader my be assigned ownership

Group: In addition to the privileges assigned to an owner, a named group of users may also be granted access rights, such that membership in the group is sufficient to exercise these access rights

World: The least amount of access is granted to users who are able to access the system but are not included in the categories owner and group for this resource

Traditional UNIX access control

- Objects have owners
- Owners have broad control over their objects
- You own new objects that you create
- The special user account “root” can act as the owner of any object
- Only root can perform certain sensitive administration operations

Filesystem access control

- Every file has an owner and a owner group
- The owner can set the permissions of a file
- A owner group allows a file to be shared among members of the same project
  - Groups are traditionally defined in /etc/group, now in an NIS or LDAP server on the network
  - The file owner specifies what the members of the group can do with the file

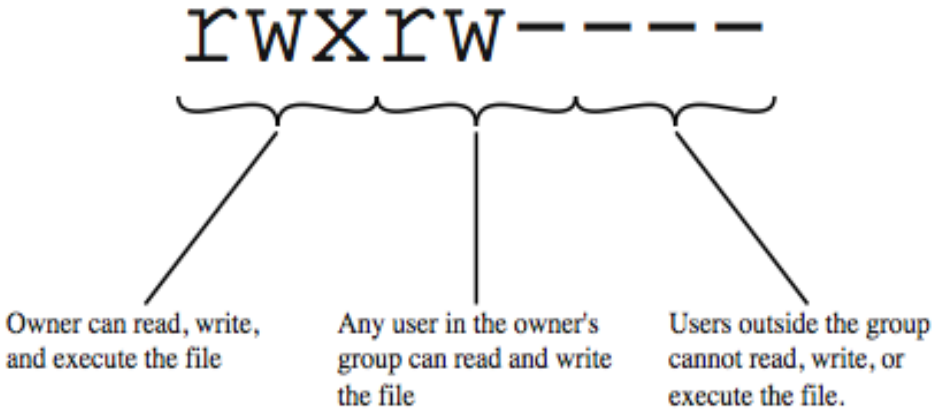
Determining file ownerships

```
$ls -l filename
-rw------- 1 yliu csstaff 4529 Jul 15 2010 todo
```

- the file todo is owned by the user yliu and the group csstaff
- Letters and dashes in the first column symbolize file permissions
  - There are 9 permission bits
  - Control who can read, write, and execute the file content
  - Also 3 other bits for executable programs (ignored for now)

The permission bits

- The 12 bits are called “mode bits”. Can be changed using “chmod” command
- 3 sets of permissions
  - Owner of the file
  - Group owners of the file
  - Everyone else
- Each set has three bits:
  - A read bit, a write bit, and an execute bit



A good reference on permissions: http://www.perlfect.com/articles/chmod.shtml

The permission bits (cont'd)

- Each user fits into only one permission set
  - Owner, group owner, or other, the most specific one
- Permissions for a file
  - Read: allow file open and read
  - Write: allow file content modification/truncation
  - Execute: allow file to be executed
- Permission for a directory
  - Read: allow content listing
  - Write: allow file creation, deletion, renaming
  - Execute: allow to enter the directory but not listing

Another example

```
$ls -l /bin/gzip
-rwxr-xr-x 3 root root 57136 Jun 15 2004 /
  bin/gzip
```

- the first character is a dash, means a regular file
- Owner has all permissions, everyone else has only read and execute permissions
- Other content: link count for the file; owner, and group owner; file size in bytes, date of last modification, file name

The *chmod* command

- Used to change the permissions on a file
- Only owner of the file and superuser can run it

Examples

- `chmod u+w todo` Adds write for the owner of file
- `ug=rw,o=r` Gives r/w to owner & group, read to others
- `a-x` Removes execute for all categories
- `g=u` Makes the group permissions the same as owner

u the owner user
g the owner group
o others (neither u, nor g)
a all users

The *chown* command

- Change a file's ownership and group ownership
  - For ownership: must be superuser
  - For group ownership: must be superuser, or both file owner and belong to target group
- Example: `chown matt:staff myfile`

  change myfile's owner to matt, owner group to staff
- There is also a `chgrp` command to change the group owner of a file only

The root account

- UNIX's omnipotent administrative user (UID 0)
- Also known as the superuser account, actual username is "root"
- Traditional UNIX allows the superuser (or any process whose effective UID is 0) to perform any valid operation on any file or process
- Restricted operations (only root can perform):

  setting hostname, system clock, configuring network, shutting down system, creating device files, change own process' UID and GID (e.g., login)

Modern access control

- Traditional one
  - simple, predictable, capable for most access control needs at the average site
  - Supported by all variants, remains the default
- Modern access control mechanisms
  - Role-based access control (RBAC)
  - SELinux: security-enhanced Linux
  - POSIX capabilities
  - Access control lists (ACLs)

Controlling processes

- Reviews: a process is an OS' abstraction for a running program (see Module 2)
- Consists of address space, user stack, and a set of data structures within the kernel (PCB)
  - address space map
  - current status (sleeping, stopped, runnable…)
  - execution priority
  - resource usage
  - files, network ports of the process
  - owner of the process

Components of a process

- PID: Process ID number
  - Unique ID assigned by the kernel
  - PIDs are assigned in order as processes are created
- PPID: parent PID
  - An existing process must clone itself to create a new process, i.e., fork(). The clone then runs a potentially different program
  - The original process is the parent. The clone is the child

The life cycle of a process

- When the system boots, the kernel creates and installs several processes
- The most notable: init, which has PID 1. It executes system's startup scripts
- All processes other than the ones the kernel creates are descendants of init
- At completion, _exit notifies the parent process or init (if parent terminated) the exit code of a child process

Process ownership

- A process' UID is the UID of the person who created it
- The owner of a process can send the process signals
- Can also reduce the process' scheduling priority
- Process "effective" UID determines its access permission