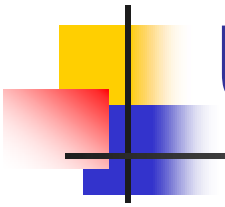
# CS590U
# Access Control: Theory and Practice

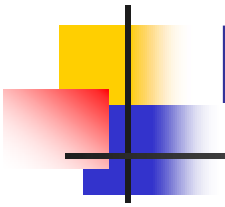
Lecture 15 (March 7)

UNIX Access Control



# Users and Groups

- Each user has a UID
- Users belong to multiple groups
  - each user has a primary group



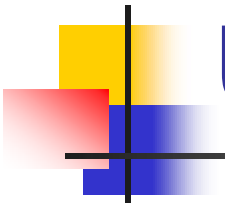
# File

- Each file has an owner and belongs to a group
  - a newly created file has either its creator's group or the directory's group
- Access rights for a file are specified for owner, group, and others

# Setuid Demystified

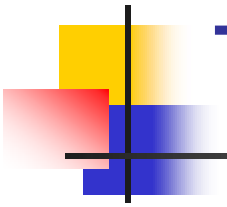
Han Chen, David Wagner, and Drew Dean

USENIX Security 2002



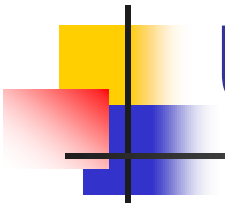
# User ID Model

- Each process has three user IDs
  - real user ID (ruid) — owner of the process
  - effective user ID (euid) — used in most access control decisions
  - saved user ID (suid)
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID



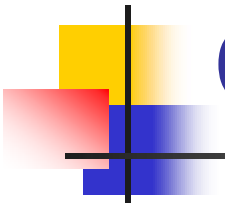
# The setuid & setgid bit of a file

- Why user-level access control is not enough?
  - system needs integrity
  - file level access control is not fine-grained enough
- The solution: Transformation Procedures
  - setuid & setgid



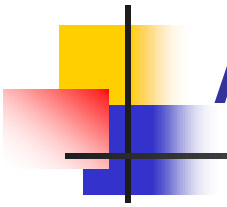
# User ID Model

- When a process is created by fork
  - it inherits all three users IDs from its parent process
- When a process executes a file by exec
  - it keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file



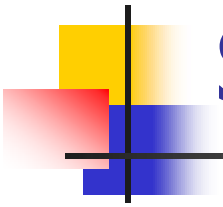
# Changing effective user IDs

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid



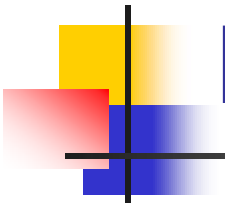
# Access Control in Early UNIX

- A process has two user IDs: real uid and effective uid and one system call setuid
- The system call setuid(id)
  - when euid is 0, setuid set both the ruid and the euid to the
  - otherwise, the setuid could only set effective uid to real uid
- A process cannot temporarily drop privilege



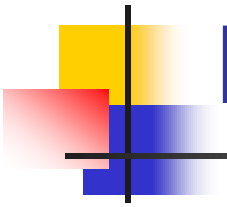
# System V

- Added saved uid & a new system call
- The system call seteuid
  - if euid is 0, seteuid could set euid to any user ID
  - otherwise, could set euid to ruid or suid
- The system call setuid is also changed
  - if euid is 0, setuid functions as seteuid
  - otherwise, setuid sets all three user IDs



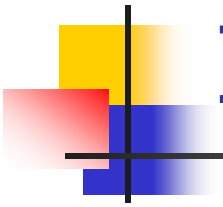
# BSD

- Uses ruid & euid, change the system call from setuid to setreuid
  - if euid is 0, then the ruid and euid could be set to any user ID
  - otherwise, either the ruid or the euid could be set to value of the other one
    - enables a process to swap ruid & euid



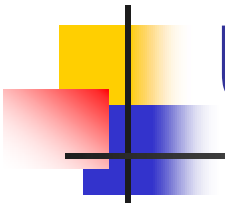
# Modern UNIX

- System V & BSD affect each other, both implemented setuid, seteuid, setreuid, with different semantics
  - some modern UNIX introduced setresuid
- Things get messy, complicated, and inconsistent, and buggy
  - POSIX standard, Solaris, FreeBSD, Linux
  - (See the paper)



# Improved API

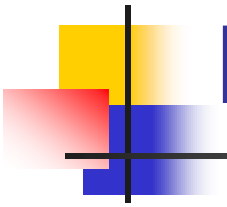
- Three method calls
  - drop_priv_temp
  - drop_priv_perm
  - restore_priv
- Morale from this?
  - mixing objectives & mechanisms
  - mechanisms got so complicated that they cannot correctly implemented and used



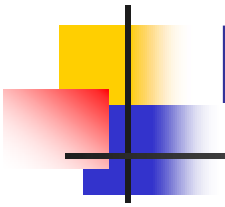
# UNIX Philosophy

- UNIX was based on a number of compact programs (tools), each of which performed a single function. Complicated tasks are performed by putting tools together.
- As tools are simple, what a tool can do depends upon who runs the tool.
  - Thus access control is by user

# Different Kinds of Programs Need Different Access Control



- Simple user tools
  - user-based access control is fine
- Deamons
  - well-defined behavior, needs some privileged access, needs to be setuid
- Complex user applications
  - browsers, email clients,



# Next Lecture

- UNIX Access Control: Process Confinement