CSci 5271 Introduction to Computer Security Day 11: OS security: higher assurance

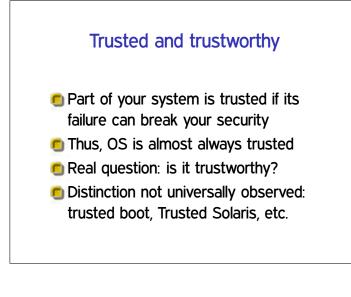
Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

OS trust and assurance

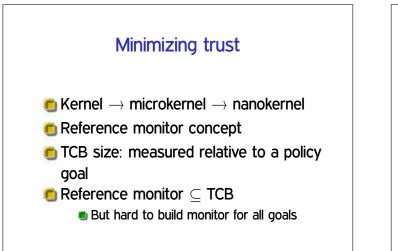
Announcements intermission

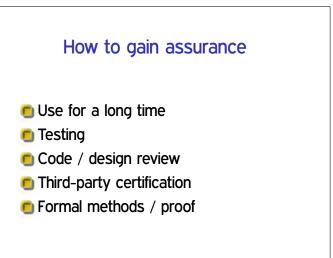
Unix access control





- How do you know you're talking to the right software?
- And no one is sniffing the data?
- 🖲 Example: Trojan login screen
 - Or worse: unlock screensaver with root password
 - Origin of "Press Ctrl-Alt-Del to log in"







Orange book OS evaluation

- Trusted Computer System Evaluation Criteria
- D. Minimal protection
- C. Discretionary protection

 C2 adds, e.g., secure audit over C1
 B. Mandatory protection
 - B1<B2<B3: stricter classic MLS</p>
- A. Verified protection



Common Criteria, Anderson's view

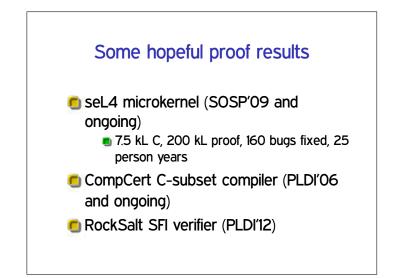
- Many profiles don't specify the right things
- OSes evaluated only in unrealistic environments
 - E.g., unpatched Windows XP with no network attacks
- Corruption, Manipulation, and Inertia
 - Pernicious innovation: evaluation paid for by vendor
 - Labs beholden to national security apparatus

Formal methods and proof

- Can math come to the rescue?
- Checking design vs. implementation
- Automation possible only with other tradeoffs
 - E.g., bounded size model
- Starting to become possible: machine-checked proof

Proof and complexity

- Formal proof is only feasible for programs that are small and elegant
- If you honestly care about assurance, you want your TCB small and elegant anyway
- Should provability further guide design?

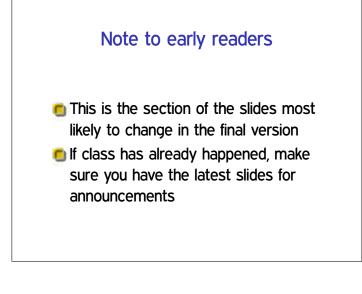


Outline

OS trust and assurance

Announcements intermission

Unix access control

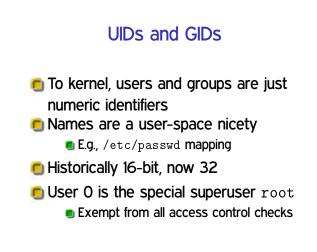


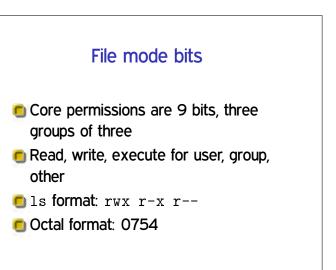
Outline

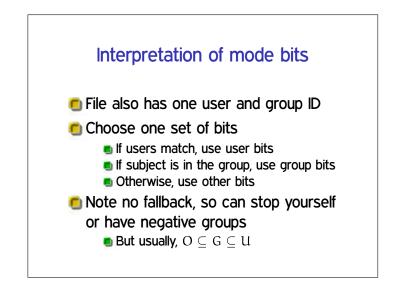
OS trust and assurance

Announcements intermission

Unix access control







Directory mode bits

- Same bits, slightly different interpretation
- 🖲 Read: list contents (e.g., 1s)
- 🖲 Write: add or delete files
- 🖲 Execute: traverse
- X but not R means: have to know the names

Process UIDs and setuid(2)

 UID is inherited by child processes, and an unprivileged process can't change it
 But there are syscalls root can use to change the UID, starting with setuid
 E.g., login program, SSH server

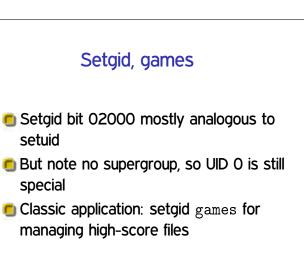
Setuid programs, different UIDs

- If 04000 "setuid" bit set, newly exec'd process will take UID of its file owner
 Other side conditions, like process not traced
- Specifically the effective UID is changed, while the real UID is unchanged

Shows who called you, allows switching back

More different UIDs Two mechanisms for temporary switching: Swap real UID and effective UID (BSD) Remember saved UID, allow switching to it (System V)

- Modern systems support both mechanisms at the same time
- Linux only: file-system UID
 - Once used for NFS servers, now mostly obsolete



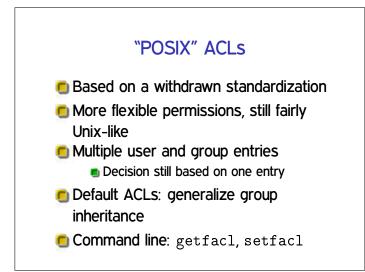
Special case: /tmp We'd like to allow anyone to make files in /tmp So, everyone should have write permission

But don't want Alice deleting Bob's files

Solution: "sticky bit" 01000

Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries with have the parent's group
 (Historic BSD behavior)
- Also, directories will themselves inherit 02000



ACL legacy interactions

- Hard problem: don't break security of legacy code

 Suggests: "fail closed"
- Contrary pressure: don't want to break functionality
 - Suggests: "fail open"
- POSIX ACL design: old group permission bits are a mask on all novel permissions

"POSIX" "capabilities"

- Divide root privilege into smaller (~35) pieces
- 🖲 Note: not real capabilities
- First runtime only, then added to FS similar to setuid
- 🖲 Motivating example: ping
- 🖲 Also allows permanent disabling

Privilege escalation dangers

- Many pieces of the root privilege are enough to regain the whole thing
 - Access to files as UID 0
 - CAP_DAC_OVERRIDE
 - CAP_FOWNER
 - CAP_SYS_MODULE
 - CAP_MKNOD
 - CAP_PTRACE
 - CAP_SYS_ADMIN (mount)

