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

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Capability-based access control

OS trust and assurance

Assignment debrief and announcements

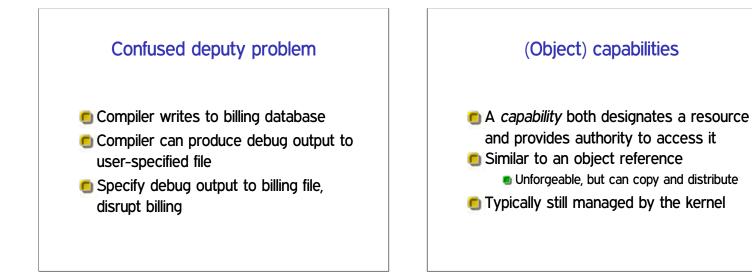
More Unix access control

ACLs: no fine-grained subjects

- Subjects are a list of usernames maintained by a sysadmin
- Unusual to have a separate subject for an application
- Cannot easily subset access (sandbox)

ACLs: ambient authority

- All authority exists by virtue of identity
- Kernel automatically applies all available authority
- Authority applied incorrectly leads to attacks



Capability slogans (Miller et al.)

- No designation with authority
- Dynamic subject creation
- Subject-aggregated authority mgmt.
- No ambient authority
- Composability of authorities
- Access-controlled delegation
- Dynamic resource creation

Partial example: Unix FDs

Authority to access a specific file
 Managed by kernel on behalf of process
 Can be passed between processes

 Though rare other than parent to child

 Unix not designed to use pervasively

Distinguish: password capabilities

- Bit pattern itself is the capability
 No centralized management
 Modern example: authorization using
- cryptographic certificates

Revocation with capabilities

- Use indirection: give real capability via a pair of middlemen
- Retain capability to tell R to drop capability to B
- Depends on composability

Confinement with capabilities

- A cannot pass a capability to B if it cannot communicate with A at all
- Disconnected parts of the capability graph cannot be reconnected
- Depends on controlled delegation and data/capability distinction

OKL4 and seL4

- Commercial and research microkernels
- Recent versions of OKL4 use capability design from seL4
- Used as a hypervisor, e.g. underneath paravirtualized Linux
- Shipped on over 1 billion cell phones

Joe-E and Caja

Dialects of Java and JavaScript (resp.) using capabilities for confined execution
 E.g., of JavaScript in an advertisement
 Note reliance on Java and JavaScript type safety

Outline

Capability-based access control

OS trust and assurance

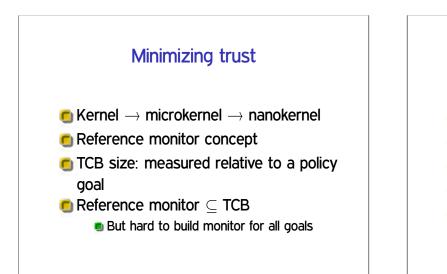
Assignment debrief and announcements

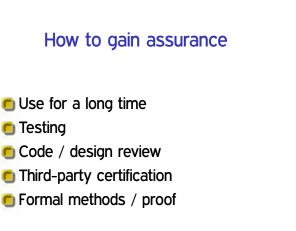
More Unix access control

Trusted and trustworthy Part of your system is trusted if its failure can break your security Thus, OS is almost always trusted Real question: is it trustworthy? Distinction not universally observed: trusted boot, Trusted Solaris, etc.

Trusted (I/O) path

- 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

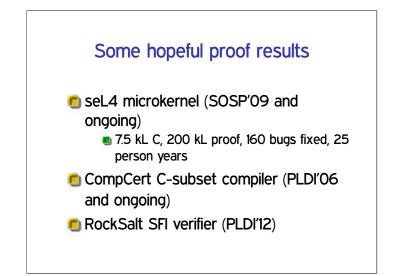
- Many profiles don't specify the right things
- OSes evaluated only in unrealistic environments
 - E.g., unpatched Windows XP with no network attacks
- Corruption, Manipulation, and Inertia"
 - Pernicious innovation: evaluation paid for by vendor
 - Labs beholden to national security apparatus

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?



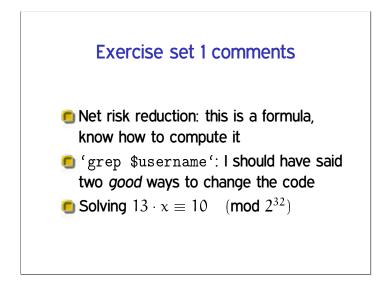
Outline

Capability-based access control

OS trust and assurance

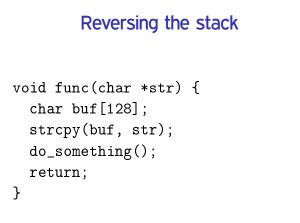
Assignment debrief and announcements

More Unix access control



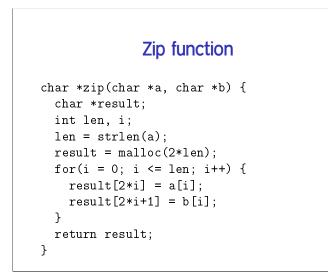
Silly function

stat(pathname, &f); stat("/what/ever", &we); if (f.st_dev == we.st_dev && f.st_ino == we.st_ino) { return; } rfd = open(pathname, O_RDONLY); buf = malloc(f.st_size - 1); read(rfd, buf, f.st_size - 1); close(rfd); stat(pathname, &f); if (f.st_dev == we.st_dev && f.st_ino == we.st_ino) { return; } wfd = open(pathname, 0_WRONLY | 0_TRUNC); write(wfd, buf, f.st_size-1); close(wfd);



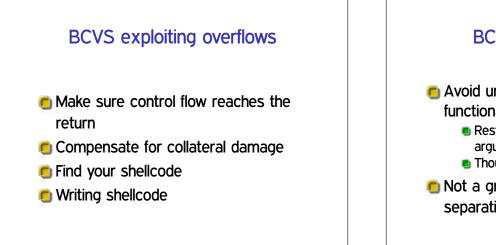


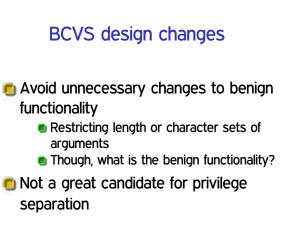


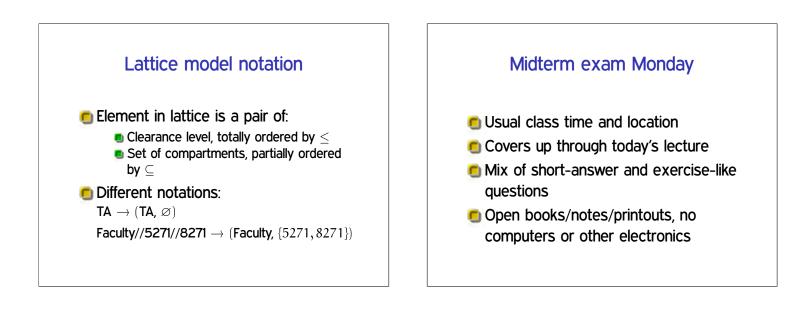


BCVS vulnerabilities

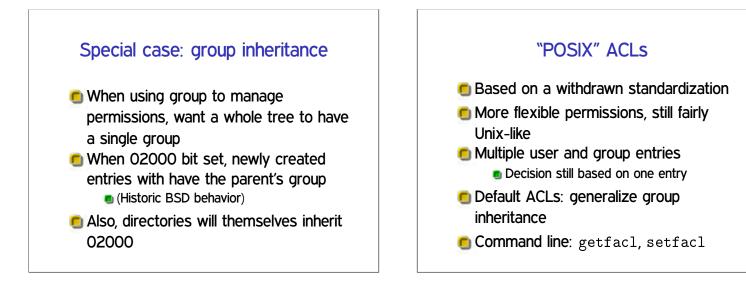
Type 1: Buffer overflows and similar
 Some easy to spot, but hard to exploit
 Type 2: Logic errors in running programs, file accesses, etc.
 Usually easier to exploit once found

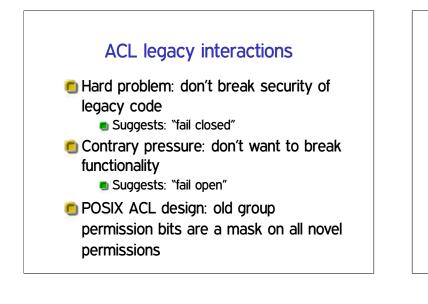






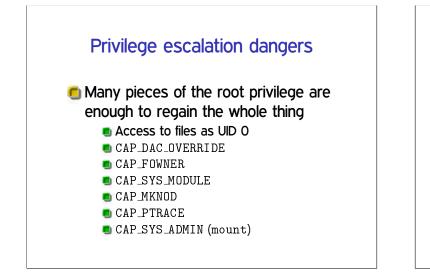






"POSIX" "capabilities"

- 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



Legacy interaction dangers

- Former bug: take away capability to drop privileges
- Use of temporary files by no-longer setuid programs
- For more details: "Exploiting capabilities", Emeric Nasi

