CSci 5271 Introduction to Computer Security Day 7: Defensive programming and design, part 1

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Saltzer & Schroeder's principles

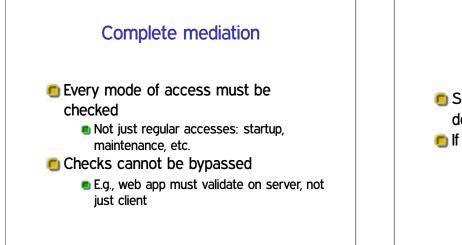
- Announcements intermission
- More secure design principles
- Software engineering for security
- Secure use of the OS

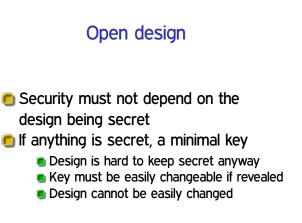
Economy of mechanism

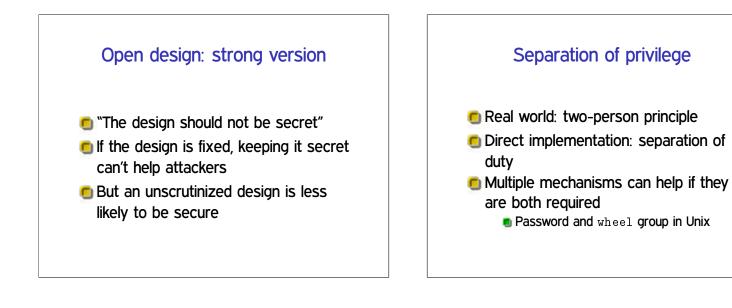
- Security mechanisms should be as simple as possible
- Good for all software, but security software needs special scrutiny

Fail-safe defaults

- When in doubt, don't give permission
- 🖲 Whitelist, don't blacklist
- 🖲 Obvious reason: if you must fail, fail safe
- 🖲 More subtle reason: incentives

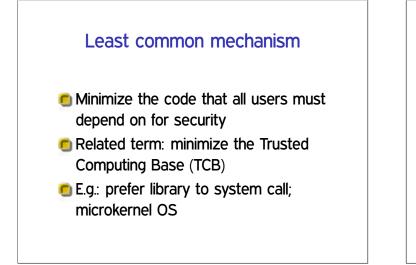






Least privilege: privilege separation

- Programs must also be divisible to avoid excess privilege
- Classic example: multi-process OpenSSH server
- **O** N.B.: Separation of privilege \neq privilege separation



Least privilege

Programs and users should have the

do their job

divisible

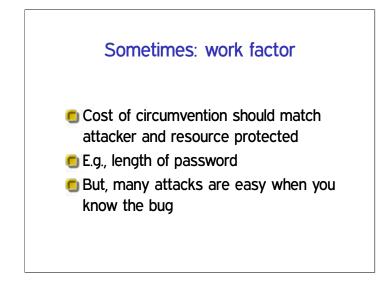
most limited set of powers needed to

Presupposes that privileges are suitably

Contrast: Unix root

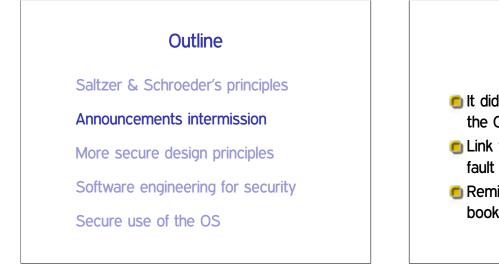
Psychological acceptability

- A system must be easy to use, if users are to apply it correctly
- Make the system's model similar to the user's mental model to minimize mistakes



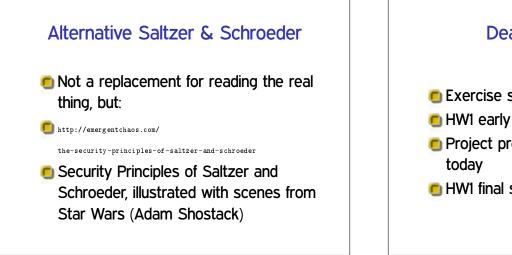
Sometimes: compromise recording

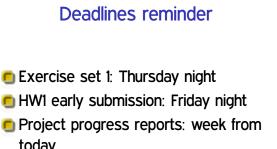
- Recording a security failure can be almost as good as preventing it
- But, few things in software can't be erased by root



Readings reminder

- It didn't seem like many people read the CFI paper...
- Link was broken until yesterday, my fault
- Reminder about library proxy bookmarklet





HW1 final submission: week from Friday

Outline

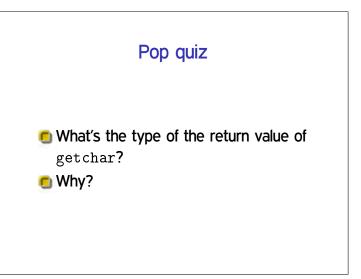
Saltzer & Schroeder's principles

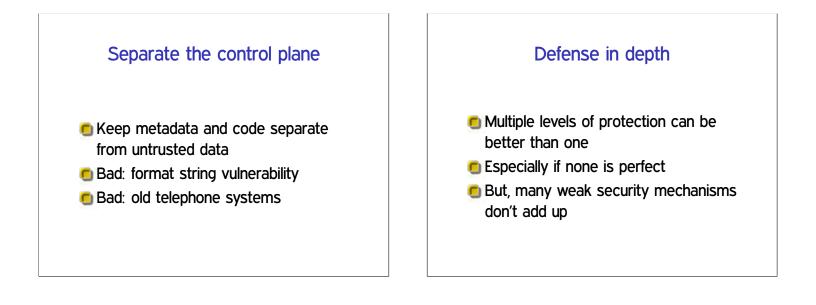
Announcements intermission

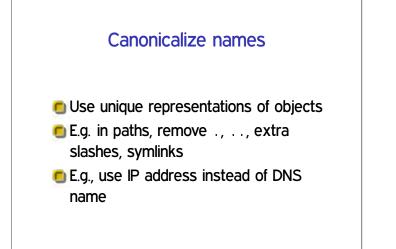
More secure design principles

Software engineering for security

Secure use of the OS









- If something goes wrong, behave in a way that's safe
- Often better to stop execution that continue in corrupted state
- E.g., better segfault that code injection

Outline

Saltzer & Schroeder's principles

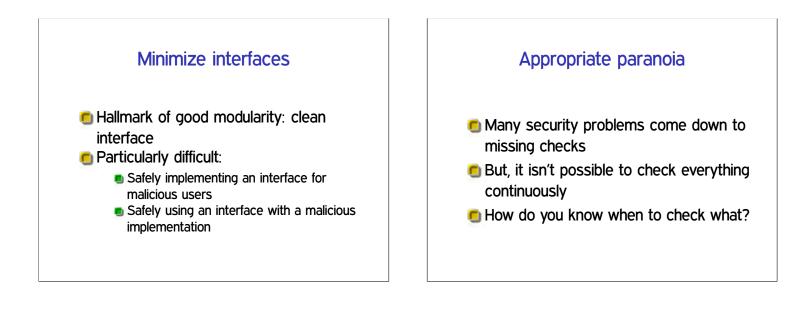
Announcements intermission

More secure design principles

Software engineering for security

Secure use of the OS

Modularity Divide software into pieces with well-defined functionality Isolate security-critical code Minimize TCB, facilitate privilege separation Improve auditability

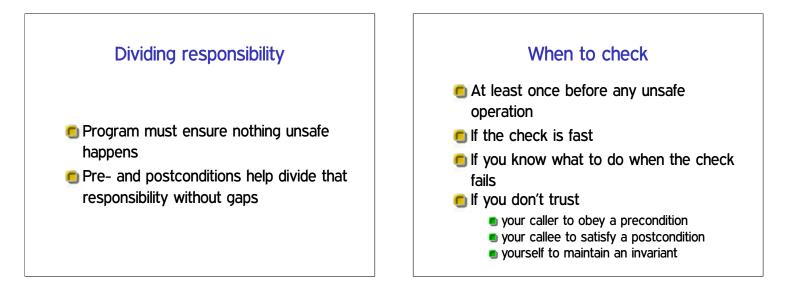


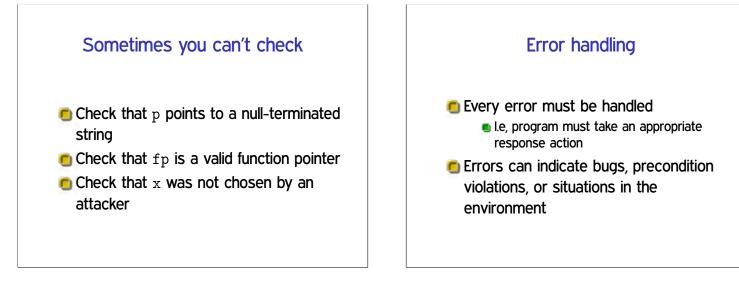
Invariant

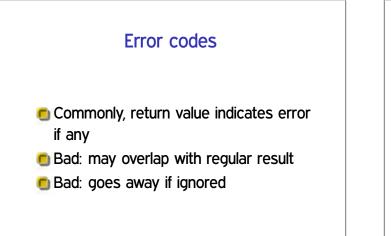
- A fact about the state of a program that should always be maintained
- Assumed in one place to guarantee in another
- Compare: proof by induction

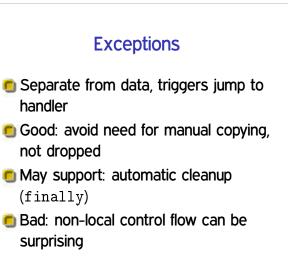
Pre- and postconditions

- Invariants before and after execution of a function
- Precondition: should be true before call
- Postcondition: should be true after return









Testing and security

- "Testing shows the presence, not the absence of bugs" – Dijkstra
- Easy versions of some bugs can be found by targeted tests:
 - Buffer overflows: long strings
 - Integer overflows: large numbers
 - Format string vulnerabilities: %x

Fuzz testing

- Random testing can also sometimes reveal bugs
- Original 'fuzz' (Miller): program </dev/urandom</p>
- Modern: small random changes to a benign input

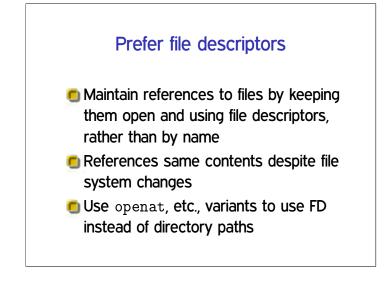


One slide on setuid/setgid

- Unix users and process have a user id number (UID) as well as one or more group IDs
- Normally, process has the IDs of the use who starts it
- A setuid program instead takes the UID of the program binary

Don't use shells or Tcl

- … in security-sensitive applications
- String interpretation and re-parsing are very hard to do safely
- Eternal Unix code bug: path names with spaces



Prefer absolute paths

- Use full paths (starting with /) for programs and files
- SPATH under local user control
- Initial working directory under local user control
 - But FD-like, so can be used in place of openat if missing

Prefer fully trusted paths

- Each directory component in a path must be write protected
- Read-only file in read-only directory can be changed if a parent directory is modified

Don't separate check from use

- 🖲 Avoid pattern of e.g., access then open
- 🖲 Instead, just handle failure of open
 - 🍯 You have to do this anyway
- Multiple references allow races
 - And access also has a history of bugs

Be careful with temporary files

 Create files exclusively with tight permissions and never reopen them
 See detailed recommendations in Wheeler
 Not quite good enough: reopen and check matching device and inode
 Fails with sufficiently patient attack

Give up privileges

- Using appropriate combinations of set*id functions

 Alas, details differ between Unix variants
 Best: give up permanently
 Second best: give up temporarily
 Detailed recommendations: Setuid
 - Demystified (USENIX'02)

