Slide 1

Administrivia

- Short Homework 8 on the Web. Nominally due Monday but will be accepted without penalty through 5pm next Wednesday.

Slide 2

Minute Essay From Last Lecture

- (Review question.)
- Obviously(?) the minimum number is no more than two, since we talked about an example deadlock scenario involving two resources.
- How about one? well, no — you can have contention for that resource, and processes might wait a long time, even possibly forever (“starvation”), but strictly speaking that’s not deadlock.

Slide 3

Security — Overview

- Goals:
  - Data confidentiality — prevent exposure of data.
  - Data integrity — prevent tampering.
  - System availability — prevent DOS (denial of service).
- What can go wrong:
  - Deliberate intrusion — from casual snooping to "serious" intrusion.
  - Accidental data loss — "acts of God", hardware or software error, human error.

Slide 4

User Authentication

- Based on "something the user knows" — e.g., passwords. Problems include where to store them, whether they can be guessed, whether they can be intercepted.
- Based on "something the user has" — e.g., key or smart card. Problems include loss/theft, forgery.
- Based on "something the user is" – biometrics. Problems include inaccuracy/spoofing.

Slide 5

Attacks From Within

- Trojan horses (and how this relates to `$PATH`).
- Login spoofing (and how this related to the Windows control-alt-delete login prompt).
- Logic bombs and trap doors.
- Buffer overflows (and how this relates to, e.g, `gets`).
- Code injection attacks.
- And many more . . .

Slide 6

Buffer Overflows

- How many times, when you read the technical description of a security flaw, do you notice the phrase “buffer overflow”? (For me — often.)
- You already know what a buffer overflow is, from writing programs in C, and how it can lead to interesting(?) bugs.
- How can this be turned to advantage by crackers? Textbook provides a brief description. A frequently-mentioned paper is called “Smashing the Stack for Fun and Profit”. Interesting reading, but the methods apparently don’t work on systems that disallow executing code from “the stack”. Textbook mentions alternatives that do still work.

Slide 7

"Attacks From Within" — Summary?

- Textbook discusses several ways programs can be made to do things their authors would not want and probably did not intend — buffer overflows, code injection attacks, etc.
- Common factor (my opinion!) is what one might call insufficient paranoia on the part of the programmers.

Slide 8

Attacks From Outside

- Can categorize as viruses (programs that reproduce themselves when run), worms (self-replicating), spyware, etc. — similar ideas, though.
- Many, many ways such code can get invoked — when legit programs are run, at boot time, when file is opened by some applications ("macro viruses"), etc.
- Also many ways it can spread — once upon a time floppies were vector of choice, now networks or e-mail. Common factors:
  - Executable content from untrustworthy source.
  - Human factors.

  "Monoculture" makes it easier!
- Virus scanners can check all executables for known viruses (exact or fuzzy matches), but hard/impossible to do this perfectly.
- Better to try to avoid viruses — some nice advice in textbook.

Slide 9

"Attacks From Outside" — Summary?

- Textbook discusses several ways "malware" (viruses, worms, etc.) can infect a system.
- Common factor (my opinion!) is allowing execution of code that does something unwanted. (Either users don't realize this is happening, or they don't realize the implications?) Social engineering is often involved. Monoculture makes the malware writer's job easier.

Slide 10

Safe Execution of "Mobile" Code

- Is there a way to safely execute code from possibly untrustworthy source? Maybe — approaches include sandboxing, interpretation, code signing.
- Example — Java's designed-in security:
  - At source level, very type-safe — no way to use `void*` pointers to access random memory. (Contrast with C!)
  - When classes are loaded, "verifier" checks for potential security problems (not generated by normal compilers, but could be done by hand).
  - At runtime, security manager controls what library routines are called — e.g., applets by default can't do file operations, many kinds of network access.

Trusted Systems

Slide 11

- Is it possible to write a secure O/S? Yes (says Tanenbaum).
- Why isn't that done?
  - People want to run existing code.
  - People prefer (or are presumed to prefer) more features to more security.

Designing a Secure System

Slide 12

- "Security through obscurity" isn't very.
- Better to give too little access than too much — give programs/people as little as will work.
- Security can't be an add-on.
- "Keep it simple, stupid."

Security — Summary

- Huge topic. Important and (I think!) interesting, though somewhat beyond the scope of this course.
- Shameless not-self-promotion: Strongly consider taking Dr. Myers's course "Information Assurance and Security" (CSCI 3311).

Slide 13

Minute Essay

- For Friday I'm tentatively planning a discussion of the boot process (as something that's relevant and interesting and not discussed yet). Other suggestions for things we didn't talk about but you'd like to?

Slide 14