

Slide 1

Administrivia

- Reminder: Homework 6 due today, Homework 7 Monday. Both accepted a little late without penalty. (But there will be a “not accepted past” date for all homeworks — probably end of reading days.)

Slide 2

Security — Overview

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

User Authentication

Slide 3

- 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.

Attacks From Within

Slide 4

- Trojan horses (and how this relates to `$PATH`).
- Login spoofing.
- Logic bombs and trap doors.
- Buffer overflows (and how this relates to, e.g., `gets`).
- And many more ...

Buffer Overflows

Slide 5

- 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? Tanenbaum provides a brief description. A frequently-mentioned paper is called “Smashing the Stack for Fun and Profit”.

Attacks From Outside

Slide 6

- Can categorize as viruses (programs that reproduce themselves when run) and worms (self-replicating) — 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 7

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.
 - 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.

Slide 8

Designing a Secure System

- “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.”

Trusted Systems

- 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.

Slide 9

Minute Essay

- Over the course of the semester I've told several "war stories" — tales of woe that taught me (or someone) something. Do you have a favorite war story to tell?

Slide 10