Lecture notes for 3.1-3.2 software vulnerabilities

Software development lifecycle (simplified)
- Requirements gathering
- Design
- Implementation
- Testing/evaluation
- Deployment
- Maintenance

Errors at many points may introduce exploitable flaws
- Design errors (windows metafile flaw)
- Testing errors (fail to identify flaw)
  - Testing normally shows that desired behaviors are present
  - Testing should also show that undesired behaviors are absent
    - Difficult/impossible to list all undesired behaviors
- Maintenance errors (patch introduces new bugs)

Software is designed by people; it will always be buggy

Common implementation errors in C software
- Buffer overflows (stack)
- Buffer overflows (heap)
- Format string vulnerabilities
- Integer overflows
- Race condition vulnerabilities / TOCTTOU

Attack goal: alter execution of software
- Commonly: exec a shell
  - Remote access, local privilege escalation
- Alter resource access (TOCTTOU)

How execute a shell?
- Mechanics: execute system call exec("/bin/sh")
  - [So how do you do that in an arbitrary program?]
- Cause program control flow to jump to point in program containing that

code
- Add that code to a program, and then jump to your added code

What can be manipulated to alter control flow?
- Control flow pointers: return addresses, function pointers, dynamic linking information

[Show stack-based buffer overflow of return address or function pointer]
[Explain heap-based overflow / heap management information]
[Show format string attack / %n format]
[Explain integer overflow / wrap memory]
[Show race condition / stat.open]

[Ask students for other types of software flaws, discuss on-the-fly]

[Ask students to suggest defenses / detections]
[Ask students to compare to non-C software]
[Given that we know how to solve the problem, why does it still exist?]