Secure Programming
CS 236
On-Line MS Program
Networks and Systems Security
Peter Reiher

Outline

- Introduction
- Principles for secure software
- Major problem areas

Example Problem Areas

- Buffer overflows
- Error handling
- Access control issues
- Race conditions
- Use of randomness
- Proper use of cryptography
- Trust
- Input verification
- Variable synchronization
- Variable initialization

Error Handling

- Error handling code often gives attackers great possibilities
- It’s rarely executed and often untested
- So it might have undetected errors
- Attackers often try to compromise systems by forcing errors

A Typical Error Handling Problem

- Not cleaning everything up
- On error conditions, some variables don’t get reset
- If error not totally fatal, program continues with old values
- Could cause security mistakes
  - E.g., not releasing privileges when you should

Some Examples

- Remote denial of service attack on Apache HTTP server due to bad error handling (2010)
- Internet Explorer arbitrary code execution flaw (2007)
  - Use-after-free bug in script error handling code

Checking Return Codes

- A generalization of error handling
- Always check return codes
- A security program manager for Microsoft said this is his biggest problem
- Very dangerous to bull ahead if it turns out your call didn’t work properly
- Example: Nagios XI didn’t check the return value of `setuid()` call, allowing privilege escalation

Access Control Issues

- Programs usually run under their user’s identity with his privileges
- Some programs get expanded privileges
  - Setuid programs in Unix, e.g.
- Poor programming here can give too much access

An Example Problem

- A program that runs setuid and allows a shell to be forked
  - Giving the caller a root environment in which to run arbitrary commands
- Buffer overflows in privileged programs usually give privileged access

A Real World Example

- `/sbin/dump` from NetBSD
- Ran setgid as group `tty`
  - To notify sysadmins of important events
  - Never dropped this privilege
- Result: `dump` would start program of user’s choice as user `tty`
  - Allowing them to interact with other user’s terminals

What To Do About This?

- Avoid running programs setuid
  - Or in other OSs’ high privilege modes
- If you must, don’t make them root-owned
  - Remember, least privilege
- Change back to the real caller as soon as you can
  - Limiting exposure
- Use virtualization to compartmentalize

Virtualization Approaches

- Run stuff in a virtual machine
  - Only giving access to safe stuff
- Hard to specify what's safe
- Hard to allow safe interactions between different VMs
- VM might not have perfect isolation