Lecture 27: Non-functional properties II

CS 5150, Spring 2022

Course reminders

- Course evaluations now open, due Fri, May 13
  - We want your constructive feedback!
    - This offering tried to combine the traditional project with material on modern, scalable tools and techniques; how useful was this?
  - Counts towards homework grade
- Final presentations
  - All times and rooms have been set
- Peer evaluations
  - Scope is most recent session (not entire semester)
  - Please leave comments

Deployment

- Client is not a fellow developer; needs to validate a *production* deployment
  - Not "click 'Run' in this IDE"
  - Not a "DEBUG" build, but a Release build
  - Not using an embedded dev server
- Client has data produced by old system
  - Data must be updated or imported
  - A "clean slate" acceptance test is not sufficient
- Internal projects: See Ed post for clarifications

Security

… continued from Lecture 26

Poll: PollEv.com/cs5150

A web service sorts user-provided data using QuickSort with median-of-3 pivoting. Uploads are limited to N bytes. What is the worst-case time complexity a user can trigger?

Availability and denial-of-service (DoS)

- Software that cannot be used is not useful
  - Even if results are correct and data is safe
- Network attacks
- Complexity attacks
  - Beware algorithms with worst case >> average case
- Compatibility
  - Beware downgrade attacks
- Avoidance & mitigation
  - Quotas & timeouts
- What is the appropriate failsafe configuration?
  - Fail-closed vs. fail-open
  - e.g. ATM vs. secure exit

Responsibility & accountability

- Software engineers and system administrators have access to highly privileged data and capabilities
  - Examples of abuse: data leaks, deliberate bugs

- Who had access to or did access certain resources?
  - Require authentication for code, config changes
  - Audit logs

Debugging features & defaults

- Often useful to bypass access control during development
  - Spoof multiple user roles for testing
  - Manipulate system at low level to diagnose bugs
- Tempting to allow easy access in production
  - Tech support, service technicians, remote patching
- Backdoor accounts, default credentials, unnecessary services are major source of vulnerabilities
  - Audit release builds for hard-coded accounts, debug-only components

IP & secrets protection

- Compiled software can be reverse-engineered
  - Strip debugging symbols for release (also saves space)
    - Save a copy internally for developers
  - Obfuscation, self-encryption can slow down analysis
  - Disable microcontroller debugging features (including flash readout)
  - Embed copyright, unique markers
  - Less of a concern for open-source software, service providers
- Protect high-value secrets (private keys, API keys)
  - Do not commit to source code repository
  - Use secure hardware modules

Trust and UI

- Users make poor security decisions
  - User interfaces (e.g. web browsers, mobile OSs) have a large impact on quality of decisions

- Consumer Reports: poor rating to any device that allows poor user security or default accounts

Safety and reliability

Terminology

- Mishap (generic): an event that is potentially unsafe
- Hazard: software exhibits unsafe behavior, but mitigation is successful
- Incident: Unsafe behavior leads to unsafe conditions, but circumstances avoided injury
- Accident: Unsafe behavior leads to injury

- Risk (review)
  - Likelihood
  - Consequence

Better Embedded System Software. Koopman 2010

Safety Integrity Levels (SIL)

- 4: Catastrophic (likely to kill people)
- 3: Critical (likely to cause injury, possibly death)
- 2: Significant (might cause injury)
- 1: Minor (contributes to unsafe conditions)
- 0: Nuisance

- Different levels target different mishap rates
  - 4: 1,000,000,000 hrs
  - 3: 10,000,000 hrs
  - 2: 100,000 hrs
  - 1: 1,000 hrs
  - 0: 100 hrs
- Testing alone cannot verify most stringent mishap rates

Software safety classes

NASA

- Class A: Human-rated flight software
- Class B
- Class C: Testing & verification of class A/B
- Class D: Engineering design
- Class E: Exploratory utilities
- Class F: Business/IT
- Class G:
- Class H: General-purpose

Medical (IEC 62304)

- Class C: Death or serious injury possible
- Class B: Non-serious injury possible
- Class A: No damage to health possible

Criticality depends on intended use!

Theme: Different projects require different development processes

- Techniques for ensuring software quality can be expensive
- Choose a process that meets the needs of the application with minimal overhead
  - But avoid a proliferation of different processes within an organization
- Example
  - Class A: Process training, ticket vetting, multiple reviewers, test coverage, ticket review
  - Class C: Ticket, one reviewer, verification evidence

Dependability terminology

- Fault: bit flip, execution of buggy code
- Failure: fault leads to incorrect computation
- Error: failure leads to observable misbehavior

- Mean Time Between Failures (MTBF): inverse of error rate
  - Assume reliability decays exponentially with time
  - After 1 MTBF, only 37% of units are still functioning without error

Hardware reliability

- Assumption of random, independent component failures
  - Serial dependencies reduce reliability
  - Redundancy increases reliability
    - Rate of *component failures* increases, rate of *system errors* decreases
- Software must contend with hardware unreliability
  - In datacenters, failures occur regularly
  - Bit flips occur in high-radiation environments
- But hardware reliability analysis is a poor fit for software
  - Violates assumption of random, independent failures
  - Analysis and mitigation techniques from hardware do not apply

Voting

- Redundancy can be used to mitigate independent failures
  - "Triple Modular Redundancy" common in space systems
- Aviation anecdotes
  - Qantas 72: Single bad sensor value used instead of two good sensor values
  - Boeing 737 MAX: Only one of two sensors used

Australian Transport Safety Bureau

Software reliability

- Bugs are not random, independent
  - Example: Ariane 5 rocket
  - Example: F-22 crossing International Date Line
- Techniques to improve software reliability
  - Improve software quality (process)
  - State scrubbing
    - Monitor health, invariants
    - Restart failed subsystems
  - Software diversity
    - Example: Space shuttle

Watchdog timers

- Hardware feature in modern processors
- Expects a periodic "still alive" message
- Reboots system if message not received in time
    - Startup runs self-tests, consistency checks, re-establishes invariants

Creating safe systems

- Creating safe systems requires analysis during requirements and system design beyond the scope of this course
  - Failure Mode and Effects Analysis (FMEA)
  - HAZOP (HAZard and OPerability)
    - "What if [requirement] is {late, more, reversed}?"
  - Fault Tree Analysis

Example: JBIG2

- How safety-critical is image compression in fax machines?
  - https://www.dkriesel.com/en/blog/2013/0802_xerox-workcentres_are_switching_written_numbers_when_scanning

- Also an example of how compatibility enlarges attack surface: https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html