WHAT CAN GO WRONG IN SOFTWARE?

- Bugs
  - Improper specification
  - Improper implementation
- Race conditions
- Holes in security
- Etc etc...
THERAC-25

- Radiation therapy machine
- Designed to target cancerous cells with X-rays or electron beam
- Killed 6 patients in 1986 and 1987
WHAT WENT WRONG?

- Previous versions of the machine had hardware interlocks to prevent operating in an invalid mode.

- Hardware locks and much of the manual controls removed in favor of software control.

- Software controls had a race condition.
  - If X-ray mode was selected, machine would begin set up (a process taking 8 seconds).
  - If switched to Electron mode during this time, the turntable (which directs the radiation) would enter an unknown state.
THERAC-25 SOFTWARE

- Entire design and code base was flawed
- No timing analysis of a real-time system
- No unit tests
- No fault tree analysis
- Written by one software engineer and no independent review
- Never tested as software-hardware combination before reaching hospital
BUT EVEN AFTER THE SOFTWARE WAS PATCHED . . .

- Another patient died due to a counter overflow that resulted in the mirror not moving correctly.
PRIUS BRAKING SYSTEM

- 2010 Prius models and Lexus hybrids recalled
- Issue with software of regenerative braking system
- Brakes temporarily stopped working when going over a bump
- Fixed with software patch
PRIUS ACCELERATION SYSTEM

- In 2009-2011 three Prius recalls because of claims that vehicle would uncontrollably accelerate
- Likely it was mechanical rather than electronic
  - Sticky accelerator pedals and floor mats that caught accelerator pedals
- In 2013 Toyota still found liable because Toyota did not follow best practices for a real-time operating system
  - No protection from cosmic ray bit flips
  - Run-time stack not large enough so possibility of data overwrite
INTEL PENTIUM FLOATING POINT DIVIDE

- Pentium floating point error discovered in 1994
- Missing entries in table for SRT division in FPU control logic leads to incorrect look up
- Only two paths to reach bug
- Intel received backlash from mainstream news sources despite claiming effects were small and unlikely
HOW BAD WAS THIS BUG?

- Hard to say since it was caught quite quickly
- Discovered by a mathematician working with distribution of primes
- In 1996 Ariane 5 rocket exploded 37 seconds after liftoff due to conversion of 64-bit floating point to a 16-bit integer
  - Overflow wasn’t handled so computer cleared memory
  - Memory dump interpreted as an instruction to rocket nozzles
In 2012 trading company lost $440M in 30 minutes

Revenue from previous quarter was $288M

Trading glitch sold overvalued shares back to market at a lower price

Affected 148 companies in the New York Stock Exchange
WHAT WENT WRONG?

- Technician did not copy new trading code (RLP) to one of eight servers that routed equity orders.
- RLP code repurposed flag formerly used to activate an old function “Power Peg.”
- Power Peg function for testing trading algorithms by moving stock prices higher and lower.
- Orders sent to eighth server with repurposed flag triggered Power Peg.
WHO IS RESPONSIBLE FOR:

- Therac-25?
- Prius acceleration/braking systems?
- Pentium FDIV?
- Knight trading glitch?

- What should happen to the responsible parties?
You have just been hired onto a drone delivery startup company. You thought you were being hired for backend web development, but it becomes clear the company is short-handed, so you are expected to help out the embedded systems team. Although you have very little experience in this area, you are eager to learn more. Between the pressing deadlines and constantly shifting milestones, you have very little mentorship or oversight. You talk to your manager to express your concerns.
HOW CAN WE PREVENT BUGS?
FORMAL VERIFICATION

- Formal verification can prove correctness of algorithms in certain circumstances
  - Model checking explores all states and transitions in the model
  - Theorem provers and satisfiability solvers can determine system guarantees based on its specifications
  - Dependentely typed programming (such as Agda) uses highly specific and expressive types to ensure code that can compile must run according to its specification
SOFTWARE SIMULATION

- Formal verification doesn’t necessarily scale to large systems and can be slow to test
  - Representing a large-scale system as a verifiable model may be difficult or impossible
- System simulation allows for testing on models that may be harder to verify mathematically
  - Helps creators reason about system’s model before and during the creation of that system
RACE CONDITIONS

- Occur when order-dependent events happen in an unexpected order
- Nondeterminism makes them difficult to reproduce
- Changes to program (such as debug mode) can introduce or eliminate unexpected program output
- Does not mean race condition is fixed!
TESTING AND EXPERIMENTATION

- Unit tests help reason about functionality and expected behavior
- Dynamic analysis can check unusual paths and states
- Quality assurance (QA) allows additional eyes to assess weaknesses in software
- But there will still be bugs!
HOW THE HEARTBLEED BUG WORKS:

SERVER: ARE YOU STILL THERE?
IF SO, REPLY "HEARTBEAT" (6 LETTERS).

SERVER: ARE YOU STILL THERE?
IF SO, REPLY "NADA" (4 LETTERS).

SERVER: ARE YOU STILL THERE?
IF SO, REPLY "WHAT" (5 LETTERS).

SERVER: ARE YOU STILL THERE?
IF SO, REPLY "WHAT" (5 LETTERS).

SERVER: ARE YOU STILL THERE?
IF SO, REPLY "WHAT" (5 LETTERS).
IS THIS PROBLEM GETTING WORSE OR BETTER?
REFERENCES