WIKIPEDIA

Therac-25

The **Therac-25** was a radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) in 1982 after the Therac-6 and Therac-20 units (the earlier units had been produced in partnership with CGR of France).

It was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation.[1]:425 Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury.[2] These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics and software engineering. Additionally the overconfidence of the engineers[1]:428 and lack of proper due diligence to resolve reported software bugs, is highlighted as an extreme case where the engineer's overconfidence in their initial work and failure to believe the end users' claims caused drastic repercussions.

Contents

Design

The machine offered two modes of radiation therapy:

- Direct electron-beam therapy, which delivered low doses of high-energy (5 MeV to 25 MeV) electrons over short periods of time;
- Megavolt X-ray therapy, which delivered X-rays produced by colliding high-energy (25 MeV) electrons into a "target".

When operating in direct electron-beam therapy mode, a low-powered electron beam was emitted directly from the machine, then spread to safe concentration using scanning magnets. When operating in megavolt X-ray mode, the machine was designed to rotate four components into the path of the electron beam: a target, which converted the electron beam into X-rays; a flattening filter, which spread the beam out over a larger area; a set of movable blocks (also called a collimator), which shaped the X-ray beam; and an X-ray ion chamber, which measured the strength of the beam.

Problem description

The accidents occurred when the high-power electron beam was activated instead of the intended low power beam, and without the beam spreader plate rotated into place. Previous models had hardware interlocks in place to prevent this, but Therac-25 had removed them, depending instead on software interlocks for safety. The software interlock could fail due to a race condition. The defect was as follows: a one-byte counter in a testing routine frequently overflowed; if an operator provided manual input to the machine at the precise moment that this counter overflowed, the interlock would fail.[2]

The high-powered electron beam struck the patients with approximately 100 times the intended dose of radiation, delivering a potentially lethal dose of beta radiation. The feeling was described by patient Ray Cox as "an intense electric shock", causing him to scream and run out of the treatment room.[3] Several days later, radiation burns appeared and the patients showed the symptoms of radiation poisoning; in three cases, the injured patients later died as a result of the overdose.[4]

Root causes

A commission concluded that the primary reason should be attributed to the bad software design and development practices, and not explicitly to several coding errors that were found. In particular, the software was designed so that it was realistically impossible to test it in a clean automated way.[5]

Researchers who investigated the accidents found several contributing causes. These included the following institutional causes:

- AECL did not have the software code independently reviewed.
- AECL did not consider the design of the software during its assessment of how the machine might produce the desired results and what failure modes existed. These form parts of the general techniques known as reliability modeling and risk management.
- The system noticed that something was wrong and halted the X-ray beam, but merely displayed the word "MALFUNCTION" followed by a number from 1 to 64. The user manual did not explain or even address the error codes, so the operator pressed the P key to override the warning and proceed anyway.
- AECL personnel, as well as machine operators, initially did not believe complaints. This was likely due to overconfidence.[1]:428
- AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

The researchers also found several engineering issues:

- The failure only occurred when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the PDP-11 computer: an "X" to (erroneously) select 25 MeV photon mode followed by "cursor up", "E" to (correctly) select 25 MeV Electron mode, then "Enter", all within eight seconds.[5] This sequence of keystrokes was improbable, and so the problem did not occur often and went unnoticed for a long time.[3]
- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
- The engineer had reused software from older models. Such methods manifest in so called Cargo coding where there is blind reliance on previously created code that is poorly understood and may or may not be applicable. These models had hardware interlocks that masked their software defects. Those hardware safeties had no way of reporting that they had been triggered, so there was no indication of the existence of faulty software commands.
- The hardware provided no way for the software to verify that sensors were working correctly (see *open-loop controller*). The table-position system was the first implicated in Therac-25's failures; the manufacturer revised it with redundant switches to cross-check their operation.
- The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly. This was missed during testing, since it took some practice before operators were able to work quickly enough to trigger this failure mode.
- The software set a flag variable by incrementing it, rather than by setting it to a fixed non-zero value. Occasionally an arithmetic overflow occurred, causing the flag to return to zero and the software to bypass safety checks.

The software was written in assembly language that might require more attention for testing and good design. However the choice of language by itself is not listed as a primary cause in the report. The machine also used its own operating system.

Leveson notes that a lesson to be drawn from the incident is to not assume that reused software is safe: "A naive assumption is often made that reusing software or using commercial off-the-shelf software will increase safety because the software will have been exercised extensively. Reusing software modules does not guarantee safety in the new system to

which they are transferred..."[5] This blind faith in poorly understood software coded paradigms is known as Cargo cult programming. In response to incidents like those associated with Therac-25, the IEC 62304 standard was created, which introduces development life cycle standards for medical device software and specific guidance on using software of unknown pedigree.[6]

See also

- IEC 62304
- Ionizing radiation
- List of civilian radiation accidents
- Nuclear and radiation accidents
- Radiation protection

Notes

1. Baase, Sara (2008). *A Gift of Fire*. Pearson Prentice Hall.
2. Leveson, Nancy G.; Turner, Clark S. (July 1993). "An Investigation of the Therac-25 Accidents" (http://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac.pdf) (PDF). *IEEE Computer*. **26** (7): 18–41.
3. Casey, Steven. *Set Phasers On Stun - Design and Human Error*. Aegean Publishing Company. pp. 11–16.
4. Rose, Barbara Wade. "Fatal Dose - Radiation Deaths linked to AECL Computer Errors" (http://www.ccnr.org/fatal_dose.html). *www.ccnr.org*. Retrieved 14 June 2016.
5. Leveson, Nancy, University of Washington (1995). "Medical Devices: The Therac-25 Accidents" (http://sunnyday.mit.edu/papers/therac.pdf) (PDF). *Safeware: System Safety, and Computers* (Update of the 1993 IEEE Computer article ed.). Addison-Wesley.
6. Hall, Ken (June 1, 2010). "Developing Medical Device Software to IEC 62304" (http://www.mddionline.com/article/developing-medical-device-software-iec-62304). *MDDI - Medical Device and Diagnostic Industry*. Retrieved 2016-12-12.

Additional reading

- Gallagher, Troy. *THERAC-25 Computerized Radiation Therapy* (https://web.archive.org/web/20071212183729/http://neptune.netcomp.monash.edu.au/cpe9001/assets/readings/www_uguelph_ca_~tgallagh_~tgallagh.html). (short summary of the Therac-25 Accidents)

- Write a paragraph about what you might have done if you were a software engineer in that project?

If I was a software engineer on the Therac-25 project, I would have been more diligent on testing edge cases. Since the problem was caused by a race condition, that means a race condition was made possible by the structure of the software. As a software engineer, we should be well-informed and aware of any implementation issues and not just the code itself. I would have been more thorough in testing. They should have trialed the implementation more with the software and not just shipped the product as is. After testing and discovering the issue, I would have notified my superiors and teammates. I would have told my superiors that we needed more time to correct the issue and retest the system again because people's lives were at stake.

- Write a paragraph about why you think Therac or VW engineers ended up doing what they did?

I think they ended up introducing a race condition because they were not thorough in testing. As stated in the article, "The lesson is to not assume reused software is safe." Since the code was written in assembly, it would have been easy to miss a race condition in the bit shifter. Especially in usability, it is hard to tell that one component that functions for one system will function the same for another system without investigating further in the code's structure. They blindly trusted the reused software would be able to just plug and chug into their system. They should have been more thorough in building the software.