« Back to Article

History's Worst Software Bugs

Simson Garfinkel 11.08.05

Last month automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly. But unlike the large-scale auto recalls of years past, the root of the Prius issue wasn't a hardware problem -- it was a programming error in the smart car's embedded code. The Prius had a software bug.

With that recall, the Prius joined the ranks of the buggy computer -- a club that began in 1945 when engineers found a moth in Panel F, Relay #70 of the Harvard Mark II system.The computer was running a test of its multiplier and adder when the engineers noticed something was wrong. The moth was trapped, removed and taped into the computer's logbook with the words: "first actual case of a bug being found."

Sixty years later, computer bugs are still with us, and show no sign of going extinct. As the line between software and hardware blurs, coding errors are increasingly playing tricks on our daily lives. Bugs don't just inhabit our operating systems and applications -- today they lurk within our cell phones and our pacemakers, our power plants and medical equipment. And now, in our cars.

But which are the worst?

It's all too easy to come up with a list of bugs that have wreaked havoc. It's harder to rate their severity. Which is worse -- a security vulnerability that's exploited by a computer worm to shut down the internet for a few days or a typo that triggers a day-long crash of the nation's phone system? The answer depends on whether you want to make a phone call or check your e-mail.

Many people believe the worst bugs are those that cause fatalities. To be sure, there haven't been many, but cases like the Therac-25 are widely seen as warnings against the widespread deployment of software in safety critical applications. Experts who study such systems, though, warn that even though the software might kill a few people, focusing on these fatalities risks inhibiting the migration of technology into areas where smarter processing is sorely needed. In the end, they say, the lack of software might kill more people than the inevitable bugs.

What seems certain is that bugs are here to stay. Here, in chronological order, is the *Wired News* list of the 10 worst software bugs of all time … so far.

July 28, 1962 -- Mariner I space probe. A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean. The investigation into the accident discovers that a formula written on paper in pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory.

1982 -- Soviet gas pipeline. Operatives working for the Central Intelligence Agency allegedly (.pdf) plant a bug in a Canadian computer system purchased to control the trans-Siberian gas pipeline. The Soviets had obtained the system as part of a wide-ranging effort to covertly purchase or steal sensitive U.S. technology. The CIA reportedly found out about the program and decided to make it backfire with equipment that would pass Soviet inspection and then fail once in operation. The resulting event is reportedly the largest non-nuclear explosion in the planet's history.

1985-1987 -- Therac-25 medical accelerator. A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had

been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

1988 -- Buffer overflow in Berkeley Unix finger daemon. The first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called *gets()* designed to get a line of text over the network. Unfortunately, *gets()* has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect.

Programmers respond by attempting to stamp out the *gets()* function in working code, but they refuse to remove it from the C programming language's standard input/output library, where it remains to this day.

1988-1996 -- Kerberos Random Number Generator. The authors of the Kerberos security system neglect to properly "seed" the program's random number generator with a truly random seed. As a result, for eight years it is possible to trivially break into any computer that relies on Kerberos for authentication. It is unknown if this bug was ever actually exploited.

January 15, 1990 -- AT&T Network Outage. A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines -- a message that the neighbors send out when they recover from a crash.

One day a switch in New York crashes and reboots, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every six seconds, leaving an estimated 60 thousand people without long distance service for nine hours. The fix: engineers load the previous software release.

1993 -- Intel Pentium floating point divide. A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range. For example, dividing 4195835.0/3145727.0 yields 1.33374 instead of 1.33382, an error of 0.006 percent. Although the bug affects few users, it becomes a public relations nightmare. With an estimated 3 million to 5 million defective chips in circulation, at first Intel only offers to replace Pentium chips for consumers who can prove that they need high accuracy; eventually the company relents and agrees to replace the chips for anyone who complains. The bug ultimately costs Intel $475 million.

1995/1996 -- The Ping of Death. A lack of sanity checks and error handling in the IP fragmentation reassembly code makes it possible to crash a wide variety of operating systems by sending a malformed "ping" packet from anywhere on the internet. Most obviously affected are computers running Windows, which lock up and display the so-called "blue screen of death" when they receive these packets. But the attack also affects many Macintosh and Unix systems as well.

June 4, 1996 -- Ariane 5 Flight 501. Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing.

First Flight 501's backup computer crashes, followed 0.05 seconds later by a crash of the primary computer. As a result of these crashed computers, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 seconds after launch.

November 2000 -- National Cancer Institute, Panama City. In a series of accidents, therapy planning software created by Multidata Systems International, a U.S. firm, miscalculates the proper dosage of radiation for patients undergoing radiation therapy.

Multidata's software allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five.

The doctors discover that they can trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors don't realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure.

At least eight patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder.

Correction:

[1] An earlier version of this story incorrectly identified the computer as the Mark 1, and the year as 1947. 11.09.05