Syrian Arab Republic

Ministry of Education

The National Center for Distinguished

12/1/2017

The 2038's Bug

A research in informatics

By: Majd Alia

Supervised by: Amjad Taha

INDEX

Introduction

We are living in an accelerating era where thinking, calculating, and planning ahead is the key to success. After the huge advance in technology we invented all kind of equipment to help the humanity to have an easier life. So we came up with machines to do some of our work for us. And the most common machine in our daily life is the computer. Therefore, hardware manufactures and software programmers keep on tracking every flow or problem that might come across their work to prevent any fatal problem from occurring. In this research I will study a bug that faces one of the most common software in machines operating, the UNIX system. I'll study the problem, discuss the effects, and predict the amount of collateral damage according to older similar problem.

In order to have a better understanding of the problem I will explain some information about:

1- The language of machines to get to the core of the problem

2- The concept of bugs to differentiate between this type of problems and other types.

3- The UNIX OS as it is the main element in this research.

4- A previous similar bug, the millennium bug as reference to predict the upcoming effects of the year 2038 bug.

AND the main question is will this bug's effects exceed our expectations, will it have massive damage upon humanity or will it be limited to miner problems….

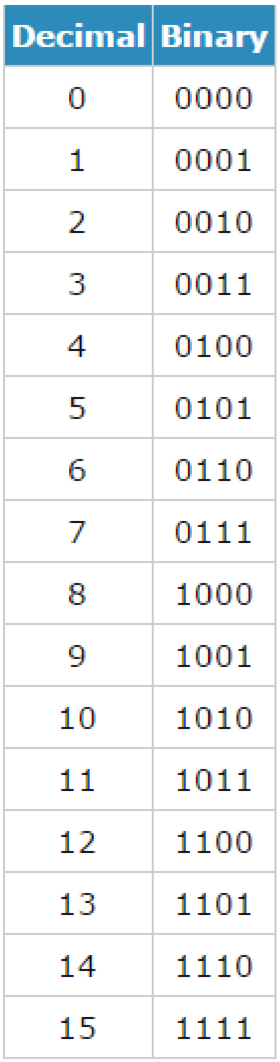
Machine language:

From the dawn of humanity, we needed ways to communicate with each other so, to fill our need we invented languages. As humans the machine needs to communicate with its hardware parts and people as well so the binary system appeared as a convenient language to communicate with the circuits and with other interfaces that can process human languages.

Why Binary?

Computers use binary numbers because they have circuits which are either on or off, which gives them two states to work from to make calculations and run processes. The two-digit, or base 2, number system is much easier for the computer to process with the circuits they have. The binary numbers have values for each space in a number, just like regular numbers with the ones, tens and hundreds places.[1]

In binary numbers, instead of each number place being multiplied by ten, they are multiplied by two. This allows long strings of number to still be translated into standard numbers and letters. Eight binary number strings are the most useful and most used set of binary numbers. The eight-number sets allow the computer to process numbers from zero to 255. These eight numbers are also the smallest binary sets that give the right amount of number options to represent letters. This is similar to how the binary numbers are used to represent pictures. Pictures are made up of pixels, which are digital representations of the picture. The pixels coincide with binary codes, which tell the computer how much red, blue or green is needed to create the color on the screen. Each pixel is normally represented by up to three bytes, or three of the eight digit binary codes.[1]



Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Figure 1 some decimal numbers and their equivalences in binary

1.https://www.reference.com/technology/computers-use-binary-numbers-153b6c083395d831 2/1/17

HOW TO CONVERT FROM DECIMAL TO BINARY:

When converting from decimal to binary the mathematical way is simplest. Start with the decimal number you want to convert and start dividing that number by two, keeping track of the remainder after each complete division. Every time you divide by two, you will divide evenly (0) or get a remainder of one (1). Following the pattern to the end, you will get a binary number. Write the remainders in the order they were generated from right to left and the result is the equivalent binary value.

Example: Convert decimal 44 to binary

1. Divide:
    - 44/2=22 reminder = 0
    - 22/2=11 reminder = 0
    - 11/2=05 reminder = 1
    - 05/2=02 reminder = 1
    - 02/2=01 reminder = 0
    - 01/2=00 reminder = 1
2. Reverse: reverse the order of reminders

The bits, in the order they were generated is 001101 Reversing the order of bits we get 101100. Properly padded with leading zeroes to fill out one byte, we get 00101100.[2]

2.www.inetdaemon.com/tutorials/basic_concepts/number_systems/binary/conversion.shtml2/1/17

UNIX Operating System:

Unix (officially UNIX) is a registered trademark of The Open Group that refers to a family of computer operating systems and tools conforming to The Open Group Base Specification, Issue 7 (also known as POSIX.1-2008 or IEEE Std 1003.1 - 2008).[3]

Proprietary Unix operating systems (and Unix-like variants) run on a wide variety of digital architectures, and are commonly used on web servers, mainframes, and supercomputers. In recent years, smartphones, tablets, and personal computers running versions or variants of Unix have become increasingly popular.[3]

The original Unix operating system was developed at AT&T's Bell Labs research center in 1969. In the 1970s and 1980s, AT&T licensed Unix to third-party vendors, leading to the development of several Unix variants, including Berkeley Unix, HP-UX, AIX, and Microsoft's Xenix. In 1993, AT&T sold the rights to the Unix operating system to Novell, Inc., which a few years later sold the Unix trademark to the consortium that eventually became The Open Group.[3]

Unix was developed using a high-level programming language (C) instead of platform-specific assembly language, enabling its portability across multiple computer platforms. Unix also was developed as a self-contained software system, comprising the operating system, development environment, utilities, documentation, and modifiable source code. These key factors led to widespread use and further development in commercial settings, and helped Unix and its variants become an important teaching and learning tool used in academic settings.[3]

There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.[4]

The UNIX operating system is made up of three parts: 1.the kernel 2.the shell and 3.the applications. [4]

3. https://kb.iu.edu/d/agat 2/1/17

4. http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html 2/1/17

Kernel: The kernel is the master control program of the operating system, handling memory management, system calls, and other low-level functions common to most programs, and providing drivers for controlling hardware.[3]

Kernel: The kernel is the master control program of the operating system, handling memory management, system calls, and other low-level functions common to most programs, and providing drivers for controlling hardware.[3]

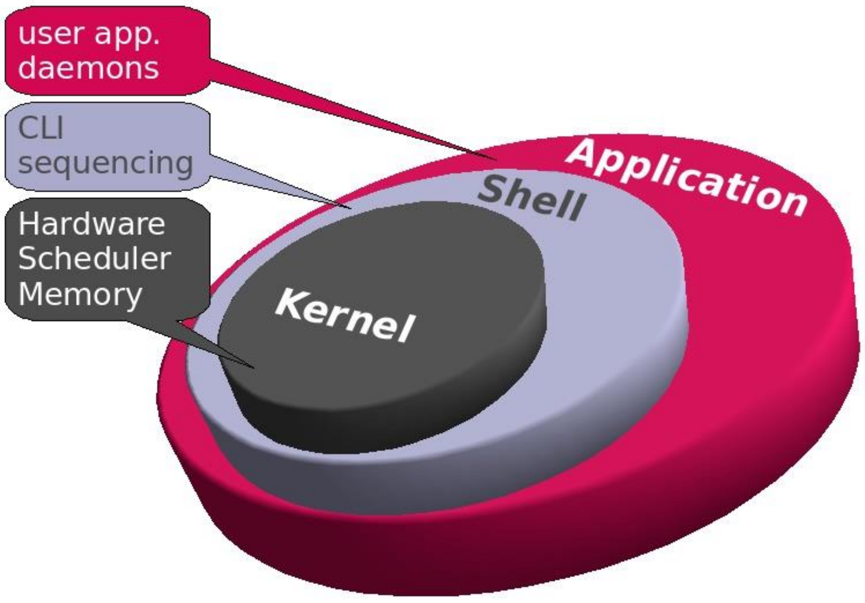


Figure 2 A diagram of the UNIX OS layers

Why UNIX is more common?

According to (adventurer.org) there are two main features in which UNIX surpasses

Other OSs like windows:

1.UNIX is more secure than Windows:

Unix was designed from the start to be a multi-user computing environment. The mechanisms for granting and denying access to files and system resources have been built into the software from the beginning.

Microsoft Windows was built to provide an entire computer to only one person, and file access control is an *ad-hoc* bolt-on which has never really worked effectively.

5.www.adventurer.org.nz/?page=writing/essays/2012-08-01_Why_we_use_Unix.txt 6/1/17

2.UNIX offers more control over your system:

When running Unix, you can have confidence that your computers are not doing things without your knowledge, like transmitting information back to corporate headquarters. Microsoft software, on the other hand, has a history of cryptic operations and has been found to report back to Microsoft on what other software you are running on your computer, violating your privacy and compromising your business information.[5]

Unix is completely transparent. All system and configuration files are readable, and the source code for all systems software is available. There is no limit to the depth of knowledge about the system you can develop, if you have the time and inclination to do so.[5]

Side-Note:

Operating systems that behave like Unix systems and provide similar utilities, but do not conform to Unix specification or are not licensed by The Open Group, are commonly known as Unix-like systems. These include a wide variety of Linux distributions (e.g., Red Hat Enterprise Linux, Ubuntu, and CentOS) and several descendants of the Berkeley Software Distribution operating system (e.g., FreeBSD, OpenBSD, and NetBSD).[3]

BUGS:

In 1946, when Hopper was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the Mark II and Mark III. Operators traced an error in the Mark II to a moth trapped in a relay, coining the term bug. This bug was carefully removed and taped to the log book. Stemming from the first bug, today we call errors or glitches in a program a bug.[6]

A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most bugs arise from mistakes and errors made in either a program's source code or its design, or in components and operating systems used by such programs. A few are caused by compilers producing incorrect code. A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy (defective).[6]

Bugs trigger errors that may have ripple effects. Bugs may have subtle effects or cause the program to crash or freeze the computer. Others qualify as security bugs and might, for example, enable a malicious user to bypass access controls in order to obtain unauthorized privileges.

In 2002, a study commissioned by the US Department of Commerce's National Institute of Standards and Technology concluded that "software bugs, or errors, are so prevalent and so detrimental that they cost the US economy an estimated $59 billion annually, or about 0.6 percent of the gross domestic product".[7]

Therefore, economist and administrations encourage programmers to check and predict the bugs in their codes to resolve them before anything bad occurs.

6. http://ei.cs.vt.edu/~history/Hopper.Danis.html 6/1/17

7.https://web.archive.org/web/20090610052743/http://www.nist.gov/public_affairs/releases/n02-10.htm 22/1/17

The millennium bug aka the Y2K bug:

Figure 3 Super computers infected by the Y2K bug

The Y2K bug was a computer flaw, or bug, that may have caused problems when dealing with dates beyond December 31, 1999. The flaw, faced by computer programmers and users all over the world on January 1, 2000, is also known as the "millennium bug." (The letter K, which stands for kilo (a unit of 1000), is commonly used to represent the number 1,000. So, Y2K stands for Year 2000.) Many skeptics believe it was barely a problem at all.

When complicated computer programs were being written during the 1960s through the 1980s, computer engineers used a two-digit code for the year. The "19" was left out. Instead of a date reading 1970, it read 70. Engineers shortened the date because data storage in computers was costly and took up a lot of space.

As the year 2000 approached, computer programmers realized that computers might not interpret 00 as 2000, but as 1900. Activities that were programmed on a daily or yearly basis would be damaged or flawed. As December 31, 1999, turned into January 1, 2000, computers might interpret December 31, 1999, turning into January 1, 1900.[8]

Banks, which calculate interest rates on a daily basis, faced real problems. Interest rates are the amount of money a lender, such as a bank, charges a customer, such as an individual or business, for a loan. Instead of the rate of interest for one day, the computer would calculate a rate of interest for minus almost 100 years!

8. http://www.nationalgeographic.org/encyclopedia/Y2K-bug 9/1/17

Centers of technology, such as power plants, were also threatened by the Y2K bug. Power plants depend on routine computer maintenance for safety checks, such as water pressure or radiation levels. Not having the correct date would throw off these calculations and possibly put nearby residents at risk.

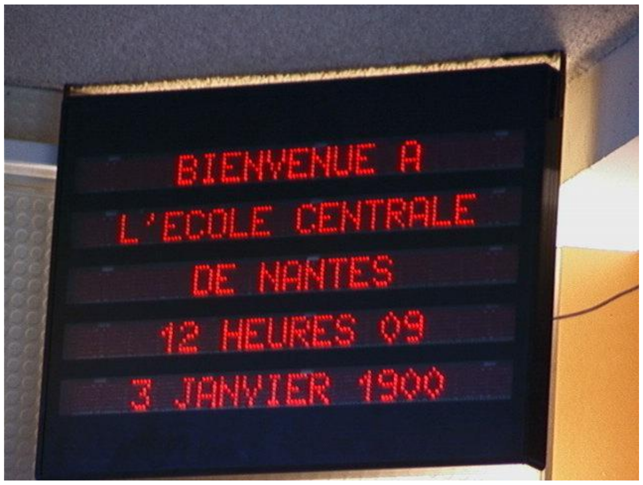


Figure 4 An electronic sign displaying the year incorrectly as 1900 on 3 January 2000 in France

Transportation also depends on the correct time and date. Airlines in particular were put at risk, as computers with records of all scheduled flights would be threatened after all, there were very few airline flights in 1900.

Y2K was both a software and hardware problem. Software refers to the electronic programs used to tell the computer what to do. Hardware is the machinery of the computer itself. Software and hardware companies raced to fix the bug and provided "Y2K compliant" programs to help. The simplest solution was the best: The date was simply expanded to a four-digit number. Governments, especially in the United States and the United Kingdom, worked to address the problem.[8]

In the end, there were very few problems. A nuclear energy facility in Ishikawa, Japan, had some of its radiation equipment fail, but backup facilities ensured there was no threat to the public. The U.S. detected missile launches in Russia and attributed that to the Y2K bug. But the missile launches were planned ahead of time as part of Russia's conflict in its republic of Chechnya. There was no computer malfunction.

Countries such as Italy, Russia, and South Korea had done little to prepare for Y2K. They had no more technological problems than those countries, like the U.S., that spent millions of dollars to combat the problem.

Due to the lack of results, many people dismissed the Y2K bug as a hoax or an end-of-the-world cult.[8]

The year 2038's bug:

Most programs written in the C programming language are relatively immune to the Y2K problem, but suffer instead from the Year 2038 problem. This problem arises because most C programs use a library of routines called the standard time library. This library establishes a standard 4-byte format for the storage of time values, and also provides a number of functions for converting, displaying and calculating time values.[9]

Y2038 refers to an issue related to the way time is handled by computers. Time is often represented as the number of seconds since Jan 1, 1970. Whenever a 32-bit signed integer is used for this, the maximum value that can be represented is +/- ~68 years, 19 days from the epoch, which corresponds to Jan 19, 2038. What happens after that is system dependent, but generally not good. A computer may act as if its time got reset to Dec 1901, or possibly to the epoch of Jan 1, 1970. It may give unexpected results or crash.[10]

Basically, any software that uses time in any way may potentially have Y2038 issues. While only a small percentage of each software component is typically affected, this is still enough to cause many problems. Embedded systems with 32-bit cores and computers running 32-bit versions of Linux and Windows and are particularly at risk.[10]

From the previous information some might say that only by switching to 64-bit cores and 64-bit OS versions we can solve the problem but unfortunately that's not even half the truth.

---

9. http://computer.howstuffworks.com/question75.htm 11/1/17

10. https://y2038.com/faq/ 11/1/17

Although virtually all new servers, desktop, and laptop computers being sold today have 64-bit hardware and operating systems. Some high-end cell phones and tablets also have 64-bit hardware and operating systems. However, the hardware and operating system are only part of the Y2038 issue. There are many other technical areas where Y2038 issues may exist, including application software, peripheral hardware, device drivers, file systems, databases, communication protocols, web content, and embedded systems. Computers with 64-bit hardware and operating systems are capable of running 32-bit software which may not be Y2038-compliant. Even 64-bit software may not be Y2038-compliant.[10]

It is also important to consider that most of the billions of embedded systems today and likely trillions that will exist by 2038 will still be using 32-bit (or less) CPUs due to factors such as power usage and the higher cost and complexity of 64-bit CPUs. Many embedded systems will not experience Y2038 issues, but a significant portion of them may.[9]

This problem is somewhat easier to fix than the Y2K problem on mainframes, fortunately. Well-written programs can simply be recompiled with a new version of the library that uses, for example, 8-byte values for the storage format. This is possible because the library encapsulates the whole time activity with its own time types and functions (unlike most mainframe programs, which did not standardize their date formats or calculations). So the Year 2038 problem should not be nearly as hard to fix as the Y2K problem was.[9]

Conclusion:

Considering the data mentioned above I believe it's fair to say that although this bug is still far to occur, communities learned their lesson from previous problems and instead of sitting still waiting for the disaster to happen like in the year 2000, they Started to seek solutions and answers for what might face them in the future.

We don't know the exact effects of this bug, but taking from what happened before, and taking in account all the insurances of related personals we can survive this problem with minor damages. Many organizations have started working towards solving this problem years ago and some of those already managed to get rid of those problems. We are prepared for most of the scenarios and with some effort there won't probably be any damages.

References

1. https://www.reference.com/technology
2. https://www.inetdaemon.com
3. https://kb.iu.edu
4. http://www.ee.surrey.ac.uk
5. https://www.adventurer.org.nz
6. http://ei.cs.vt.edu
7. https://web.archive.org
8. http://www.nationalgeographic.org
9. http://computer.howstuffworks.com
10. https://y2038.com/faq

Figures References