# Y2K38: The Bug

**Vishal Singh, Prerna Chaudhary**

***Abstract: Digital world has been threatened by many bugs but only a few seemed to pose a great danger. The bug that got most famous was Y2K. Somehow, we got over it, then, there was Y2K10. It too was resolved and now we have Y2K38. The Y2K38 bug, if not resolved, will make sure that the predictions that were made for the Y2K bug come true this time. Y2K38 bug will affect all the systems, applications and most of the embedded systems which use signed 32 bit format for representing the internal time. As the epoch for most these systems is 1, January, 1970 and since then the number of seconds which can be represented using this signed 32 bit format is 2,147,483,647 which will be equal to the time 19, January, 2038 at 03:14:07 UTC(Coordinated Universal Time). After this moment the systems will stop working correctly. There have been some solutions for this problem but a universal solution is yet to be found. All the solutions tend to delay this problem so that we can have some more time to find a good and universal solution and so does our proposed solution.***

***Index Terms-signed 32 bit integer, time_t, Y2K, Y2K38.***

## I. INTRODUCTION

Y2K bug was the result of practice of abbreviating a four digit year to two digits. The practice of representing years in two digits creates a problem when one rolls over from x99 to x00. Back in 2000, this was the reason which created many date related processing to operate incorrectly for dates and times on and after 1, January, 2000[1]. However, the number of computer failures due to Y2K is not known. This may be due to the remedial work done or because of the reticence of the organizations to report problems[1][2]. Another bug which is threatening to cause great damage is the Y2K38 bug.

The Y2K38 problems arise because of the similar reason which was responsible for the Y2K problems. As the Y2K problems resulted from not allocating enough bits for the representation of year, the Y2K38 problems are a result of not allocating enough bits for the representation of internal time. C and C++ programs use a time_t data type for the representation of dates an time internally. And 4 bytes are allocated for this representation. In these 32 bits, one bit is assigned for showing the positive/ negative integer value and hence, 31 bits are left for representing the time value. The highest value that can be represented using these 31 bits is 2,147,483,647 and what this number will represent in time_t is equal to 19 January, 2038 at 03:14:07 UTC. At this moment, every time_t used in a 32 bit C or C++ program will reach its upper limit and after that moment, the 32 bit clocks will overflow and will return some erroneous value.

**Manuscript Received on October, 2012.**

**Vishal Singh,** B.Tech.(I.T., IV Year), Meerut Institute of Engineering and Technology, Meerut, India.

**Prerna Chaudhary, ,** B.Tech.(I.T., IV Year), Meerut Institute of Engineering and Technology, Meerut, India.

January 19, 2038 is a date which will result in the failure of many computer platforms, softwares and embedded systems as these systems will run out of time.

On 19, January, 2038, as soon as the clock will strike 03:14:07, all these systems will stop working properly. And the result will be massive power outages, hospital life support system failures, bank problems, satellites falling out of orbit, et cetera. All this will happen with the softwares and systems that store system time as a signed32 bit integer and this number is taken as the number of seconds since 00:00:00 UTC(Coordinated Universal Time) on Thursday, 1 January, 1970[3],[4]. So, using signed 32 bit integer, the furthest time which can be represented is 03:14:07 UTC on Tuesday, 19 January, 2038[3],[5]. After this moment, the time will wrap around and be stored internally as a negative number which will be interpreted by such systems as a date in 1901 rather than 2038. The Y2K38 bug is also called the Unix Millennium bug because most 32-bit Unix-like systems store and manipulate time in this format and this is usually called Unix time. But this does not mean that other operating systems will not be affected from this bug, they will be as much vulnerable to this problem if they are using such a time format. Most programs which are written in the C programming language will suffer from this problem as C programs use a library of routines called the standard time library (time.h) and this library uses a standard 4-byte format for the storage of time values and hence, the maximum value that can be represented using the 4-byte format is 2,147,483,647 which is equal to 19 January, 2038 at 03:14:07 UTC. And C is the mostly used programming language in the embedded softwares, thus, making the Y2K38 bug even more dangerous as embedded systems are being used everywhere regardless of the field, be it medical, banking, electronics or any other field. All the dire predictions which were made for the Y2K bug might not have occurred then but the same problems and even worse will be faced by the world in 2038 if the required corrective and preventive measures will not be taken in time.

## II. Y2K38: THE BUG

This erroneous value may be 1, January, 1970 or 13, December, 1901 which is a Friday and hence, y2k38 bug is also called Friday, the 13th bug. The standard time library(time.h) used by the C programs, not only provides this format for storing time but it also provides certain functions for converting, displaying and calculating time values. So, all these functions when used will give erroneous results. Also, C is the most widely used programming language due to its compactness and embedded softwares mainly use C language due to which the fields in which such softwares are being used will also be adversely affected. Moreover, there are many protocols which specify 32 bit timestamp as part of their inner workings. And, hence, these protocols too will be affected from this bug.

### *A. Time and Date Representation*

Time_t is actually an integer which is counting the number of seconds that have passed since January 1, 1970 at 00:00:00 UTC. That is, a time_t value of 0 would give 00:00:00(midnight) 1, January, 1970, a time_t value which is equal to 1 would give 00:00:01(just a second after midnight) 1 January, 1970 and so on. Also one year has around 31,536,000 seconds, thus, a time_t value of 31,536,000 would return 1, January, 1971 and a value of 63,072,000 would represent 1, January, 1972.

Some times and their time_t representation

| Date & Time | Representation |
|---|---|
| l-Jan-1970, 12:00:00 AM GMT | 0 |
| l-Jan-1970, 12:00:01 AM GMT | 1 |
| l-Jan-1970, 12:01:00 AM GMT | 60 |
| l-Jan-1970, 01:00:00 AM GMT | 3600 |
| 2-Jan-1970, 12:00:00 AM GMT | 86400 |
| 3-Jan-1970, 12:00:00AM GMT | 172800 |
| l-Feb-1970, 12:00:00AM GMT | 2678400 |
| l-Jan-1971, 12:00:00AM GMT | 31 536000 |
| l-Jan-2038, 12:00:00AM GMT | 2145916800 |
| 19-Jan-2038, 12:00:00AM GMT | 2147483647 |

The time_t value of 2147483647 is the maximum no. of seconds that could be represented by time_t since, 1, January, 1970 and after this second, i.e. at -2147483648, the time-t would have a negative value and now it will calculate the time by subtracting as many seconds from 1, January, 1970 and this will give a date and time due to which many applications will fail. And the value of this time_t will be Friday, 13th December, 1901. This date is called the 'wrap around' date.

There are many data structures which will result in this problem. Infact, to list all of them is not possible but some of the data structures are-

- File systems (many file systems use only 32 bits to represent times in inode)
- databases (that have 32-bit time fields)
- embedded factory, refinery control and monitoring subsystems
- assorted medical devices
- assorted military devices

Each of the above places where data structures using 32 bit time are placed has its own risks related to failure of the product to perform as designed [3].

This upper limit of time_t will vary according to various time zones. This upper limit for different zones will be [3],[13],[14]

| | |
|---|---|
| Auckland | 19 January 2038, 16:14:07 NZDT(UTC+13:00) |
| Sydney | 19 January 2038, 14:14:07 AEDT (UTC+11:00) |
| Tokyo | 19 January 2038, 12:14:07 JST (UTC+09:00) |
| Beijing | 19 January 2038, 11:14:07 CST (UTC+08:00) |
| Mumbai | 19 January 2038, 08:44:07 IST (UTC+05:30) |
| Dubai | 19 January 2038, 07:14:07 GST (UTC+04:00) |
| Nairobi | 19 January 2038, 06:14:07 EAT (UTC+03:00) |
| Cairo | 19 January 2038, 05:14:07 EET (UTC+02:00) |
| Paris | 19 January 2038, 04:14:07 CET (UTC+01:00) |
| London | 19 January 2038, 03:14:07 GMT (UTC±00:00) |
| Brasília | 19 January 2038, 01:14:07 BRST (UTC−02:00) |
| Atlantic Time | 18 January 2038, 23:14:07 AST (UTC−04:00) |
| Eastern Time | 18 January 2038, 22:14:07 EST (UTC−05:00) |
| Central Time | 18 January 2038, 21:14:07 CST (UTC−06:00) |
| Mountain Time | 18 January 2038, 20:14:07 MST (UTC−07:00) |
| Pacific Time | 18 January 2038, 19:14:07 PST (UTC−08:00) |
| Hawaii | 18 January 2038, 17:14:07 HST (UTC−10:00) |

Below is a piece of code that will simulate the bug Y2K38 [6],[8],[9],[15]

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
int main (int argc, char **argv)
{
    time_t t;
    t = (time_t) 1000000000;
    printf ("%d, %s", (int) t, asctime (gmtime (&t)));
    t = (time_t) (0x7FFFFFFF);
    printf ("%d, %s", (int) t, asctime (gmtime (&t)));
    t++;
    printf ("%d, %s", (int) t, asctime (gmtime (&t)));

return 0;
}
```

Output-

```
1000000000, Sun Sep 9 01:46:40 20012147483647,
Tue Jan 19 03:14:07 2038-2147483648,
Fri Dec 13 20:45:52 1901
```

The output of the above program shows what will happen in 2038. All the time related functions, programs will give erroneous result. All the variable quantities are studied and measured with respect to time as it is considered to be the independent variable in most computer applications so it becomes important for this time_t variable to return correct and exact value.

### *B. Consequences of Y2K38 Problem*

The problem with Y2K38 problem is that it is not being taken that seriously and sincerely. Though we still have some 25 years for this

problem but still this problem is bigger than the y2k problem. The Y2K problem was given so much media attention that everyone knew about it but this bug is not known to much and poses bigger threats than Y2K. The use of computers has increased by leaps and bounds since 2000 and thus, the y2k38 bug will affect a much larger number of systems. Also the softwares that work on future dates will be affected by y2k38 much sooner. For example, a program that works with dates 10 years in future will have to be fixed by 2028. There are large numbers of machines which will suffer from the consequences of this problem. Many of these machines will, hopefully, be decommissioned by the year 2038 but there may be some machines which will be operating in 2038 also. These may include embedded in traffic light controllers, navigation systems, emulators, etc.

The most widely affected systems will be those which have C at their hearts, i.e. which use C as their main programming language. Many operating systems are written in C and hence, all such things which utilize time related functions in C will stop giving correct results.

Satellites falling from their orbits, life support systems failure, power grid failures all these problems are likely to occur due to 2038 bug. The basic system timekeeping functions from which most other time and date information is derived is involved in the 2038 problem which makes this problem quite serious.

## III. RELATED WORK

The related work done for the Y2K and Y2K38 bugs is:

### *A. For Y2K bug there were many solutions. Few of them are-*

1. Date Expansion- Inclusion of the century in the two digit representation was done in programs, files and databases [1].
2. Windowing- Two-digit years were retained, and programs determined the century value only when needed for particular functions, such as date comparisons and calculations. (The century "window" refers to the 100-year period to which a date belongs.) This technique, which required installing small patches of code into programs, was simpler to test and implement than date expansion [3],[7].
3. Date Re-partitioning- In legacy databases whose size could not be economically changed, six-digit year/month/day codes were converted to three-digit years (with 1999 represented as 099 and 2001 represented as 101, etc.) and three-digit days (ordinal date in year). Only input and output instructions for the date fields had to be modified, but most other date operations, and whole record operations required no change. This delays the eventual roll-over problem to the end of the year 2899 [1].

### *B. For Y2K38 bug, we don't have any universal solution yet. But some of the proposed solutions are-*

1. Change the definition of time_t- This would result in code compatibility problems as there are certain applications which are dependent on the signed 32 bit representation of time. For example if time_t is changed to an unsigned 32 bit integer then, the applications that retrieve, store or manipulate the dates prior to 1970 will not be able to do so correctly. Though doing so will delay this problem to 2106 [3].
2. Shift from 32 bit systems to 64 bit systems- Shifting from 32 bit systems to 64 bit systems gives us a new wrap around date which is over 20 times greater than the estimated age of universe i.e. about 292 billion years from now.
3. NetBSD's solution- Starting with its 6$^{th}$ major release, NetBSD is using a 64 bit time_t for both 32 bit and 64 bit architectures which supports 32 bit time_t in applications compiled for a former NetBSD release via its binary compatibility layer [3].
4. Possible solution for Linux that can be implemented [10].
5. Use of a header file- This solution states that if we use a header file given by Paul Sheer [11] instead of the header files being used [12].

## IV. SUGGESTED WORK

We want to suggest that if we can slow down the time of these 32 bit systems then we will have some more time to find a universal solution or might be able to replace most of the 32 bit systems with 64 bit systems in that extra interval of time. Now, we have 60 seconds in 1 minute but if we can allot 120 seconds to 1 minute then we will have twice as much of time we have now left for this problem. And by saying allotting 120 seconds to 1 minute, we actually mean to increase the time interval between 2 seconds for such systems. Just making that time interval twice as it is now.

With unadulterated unix time_t stamp;

1 minute=60 seconds (for both, the real world and the internal time of the 32 bit systems)

2 minutes=120 seconds

1 hour=3600 seconds

1 year=31536000 seconds

With possible time delay increase;

1 minute=120 seconds (for the real world 120 seconds will be equal to 2 minutes but these systems will take 120 seconds to internally represent passage of a single minute i.e. a delay of 60 seconds)

So, for the systems

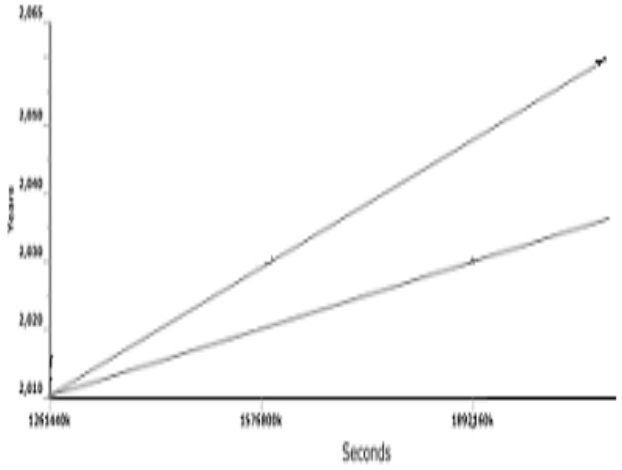
120 seconds=60 seconds;

2 hours=1 hour;

24 hours=12 hours;

2 years=1 year;

which means if this can be done, then, we will get 25 more years to resolve the Y2K38 problem. Now, we have around 25 years till 2038 bug and by doing as suggested we will have around 50 years and by that time we will find a universal solution to this problem.




The given graph between the seconds and the corresponding years they represent with ongoing internal time and the delayed internal time clearly shows that we can get extra time to resolve this problem.

## V. CONCLUSION

Y2K38 being a serious problem needs to be solved properly and within time. But, a solution that can be applied universally is not yet there. Most of the solutions are either for a particular software, system or for delaying this problem. Our suggested solution too tries to delay the problem so that we can have some extra time. Using the given solution, the year 2038 problem will become the year 2063 problem as 2, 147,483,647 would no more represent a date in the year 2038 but a date in the year 2063. However, due to our academics we were not able to some more research about the problem and its solution but in future whenever we will get time we will definitely continue our research.

## VI. ACKNOWLEDGMENT

We would like to express our sincere thanks to Dr. Nitin Goyal (Faculty, Computer Science Engineering, M.I.E.T., Meerut) and Mr. Ankur Srivastava(Faculty, Information Technology, M.I.E.T., Meerut) who were abundantly elpful and supported us throughout this research work.

## REFERENCES

1. http://en.wikipedia.org/wiki/Year_2000_problem
2. Carrington, Damian( 4 January 2000). “Was Y2K bug a boost?” BBC News. Archived from the original on 22 April 2004. Retrieved 19 September 2009
3. http://en.wikipedia.org/wiki/Year_2038_problem
4. “The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition. IEEE and The Open Group. The Open Group.2004. Retrieved 7 March 2008
5. Diomidis Spinellis(2006). Code quality: the open source perspective. Effective software development series in Safari Books Online (illustrated ed.). Adobe Press. ISBN 0-321-16607-8.
6. http://www.codeproject.com/Articles/25848/The-Year-2038-Bug-Y2K38-Problem-Many-of-your-appli
7. “The Case for Windowing: Techniques That Buy 60 Years”, article by Raymond B. Howard, Year/2000 Journal, Mar/Apr 1998.
8. http://www.ruchitsurati.net/index.php/2007/08/19/the-year-2038-bug-y2k38-problem-many-of-your-applications-will-crash/
9. http://www.2038bug.com/demo.html
10. http://linuxfinances.info/info/unix2038.html
11. http://www.2038bug.com/pivotal_gmtime_r.c.html
12. http://www.idrbt.ac.in/publications/workingpapers/Working%20Paper % 20No.%2009.pdf
13. http://en.wikipedia.org/wiki/Y2K38
14. http://en.wikipedia.org/wiki/Year_2039_problem
15. http://www.slideshare.net/ajayspi/y2-k38