THE MILLENNIUM BUG: THE PROBLEM AND THE SOLUTION

By

Isaac Tanam

B Eng M Eng PGDC CCP MNSE MCOAN

ECWA Information & Computer Science Institute, Jos

Presented at the Sensitisation Workshop for Computer Professional in Plateau State, May 1998

Introduction:

The year 2000 has been saddled with a lot of expectations. It looks as if life would start afresh in that year. There has been lots of talk about the year 2000. Everybody, including Engineers and governments, wants certain things done by the year 2000. We have heard slogans like, "Water for all by the year 2000", "Shelter for all by the year 2000", "Food for all by the year 2000", and "Health for all by the year 2000".

Some of these statements, as unrealistic as they may be, are the desires of most people, especially in developing countries like ours. Today, we have yet another of such slogans that has been bothering the minds many, especially those in the Information Technology world. It is not one of those desired by anybody, yet it is more realistic and certain. It is "**bug for all by the year 2000."**

Yes, “bug for all by the year 2000.” All may not have water, shelter, food, nor health at the turn of the year 2000, but the millennium bug is one that would affect all, directly or indirectly, come January 1, 2000.

The Genesis of the Problem:

Computers, as we know them today, are made up of the hardware and the software. The hardware consists of all the physical parts of the computer, the monitor, the keyboard and the "CPU", while the software are the instructions (programs) that make the computer work. Some software are built into the hardware, in which case they are referred to as Firmware, while some are installed after purchase and can be replaced at will. In the past, software engineers had the difficult task of developing

programs to meet up with the advancements in hardware. Today, hardware engineers are finding it increasing difficult to produce hardware that would match the sophistication of software being produced.

In the days of hardware advancements, two parts of the computer were of major concern to software engineers, the memory and the permanent storage capacity. Of these, the memory was of greater importance because every program that runs on the computer had to be in the memory, immediately or eventually. Memories at that time, were in the range less than 50KB and almost 100KB, and were very expensive. It was considered a luxury to have a computer with memory as large as 100KB. Programmers therefore had to find ways to manage the available memory. The need for efficient memory usage is what has led to the cry today about the millennium bug.

A software bug is a piece code or statement in a program that causes the program not to function properly, that is, producing wrong results, from the stand point of the user. Most bugs, like virus creation, are innocently introduced into programs. Two illustrations would make this clearer. These illustrations are in the C language.

1.
```
#include<stdio.h>
void main()
{
int amount_paid, previous_balance = 100,new_balance;
fprintf(stderr,"Enter payment>");
fscanf(stdin,"%d",&amount_paid);
new_balance = previous balance + amount paid;
fprintf(stderr,"\n Current Balance is N%d",new_balance);
}
```

This small program is expected to keep track of customers' account in a small business. The use of the keyword "**int**" for variables declaration in the fourth line assumes that no customer would make a payment of Kobos. This means that payments such as ₦5.50 or ₦90.40 is not expected. This is because **int**, in C language, only handles integers (whole numbers). **int** further assumes that no customer would make a payment, or have a balance greater than ₦32,567,

because that is the biggest number that **int** can handle. These assumptions leave two bugs in the program that would never be noticed for as long as the conditions are met. These assumptions are not realistic as it is difficult to do small financial transaction without fractions of a Naira. Also, one cannot guarantee that all payments or balances would be less than ₦32,567. To solve this problem we do not need to change the computer running this program but to change the program by replacing the **int** with some other variable type, if we could lay our hands on the source code.

2.
```
#include<stdio.h>
void main()
{
char name[15]
printf("Enter your name>");
gets(name);
printf("\n Your name is %s",name);
}
```

This small program is designed to ask for somebody's name and print out the name when typed in. It is designed to handle only names less than fourteen characters in length. If I typed in "Harrieth Tanam" (fourteen characters including space), I don't have any problem, the program prints it out. But suppose I typed "Babatunde P. Jacob" (seventeen characters including spaces), I might have a problem because the extra characters typed might wipe off some useful data in the memory to create room for themselves. The result is unpredictable. The bug in this program would never be noticed so long as nobody with names longer than fourteen characters is entered.

To solve this problem, I have to change the 15 in char name[15] to something bigger to accommodate longer names. This, of course, just like in the case of **int**, would mean using up more memory space if available. Solution of the two problems above assumes I can lay my hands on the source code (the

programs the way they are shown above) of the programs, otherwise, one has to rewrite the entire program. Re-writing these programs is easy because they contain just 8 or 9 lines. Large programs contain tens or hundreds of thousand lines. Some even have over a million lines. Those would be very difficult and expensive to re-write. It must be noted here that almost all, if not all software, no matter how good or sophisticated they may be, have one or more bugs. How soon these bugs are detected depends on the experience the one using the programs, and the extent to which he can explore the program. The programmer would always be very glad to receive notification of such bugs from you in order to make corrections to his program.

The Millennium Bug:

The millennium bug is a bug in some programs developed some years back. It must be emphasised here that this is more a software problem than a hardware problem. This bug has to do with the way dates are represented in these programs. Some of these programs were written probably between 1950's or 1980's. At this time, computer memories were still very small, but very expensive. So for proper management, programmers used two digits to represent the year in a date. For instance, November 30, 1998 would be represented as 11/30/98 (using MM/DD/YY format). This assumes that the millennium and the century in which the program is used is known. Of course, it is known to the user but not to the computer running the program. In the same way, it would be difficult to determine if a date like 1/1/00 refers to January 1, 1900 or January 1, 2000. And that is the bug (called millennium bug) in the program.

The millennium bug has always been there but never noticed and even now the effect is not seen yet. It has a definite date and time when the effect would begin to

manifest, and that time is midnight of December 31, 1999. It is for this reason that some people call the bug, "millennium time bomb".

Effects of the Millennium Bug:

The effect of this bug could range from "negligible" errors in employee's age as calculated by the computer, to as serious problems as organizations loosing or gaining huge sums of money, or even lose of lives. A few examples would buttress this.

1. Date of Employment:

Suppose a staff was employed on 11/7/60. By 11/7/00 (i.e. ll/7/2000), he would have worked for 40 years but the computer would report it as -60 years which doesn't make sense as nobody works for negative number of years. If we are to take the absolute value, then he would have worked for 60 years, even this too is wrong. So you would have errors in determining his entitlement on retirement.

2. Accounting Department:

Accounting departments (and other financial institutions) might not have any problem till 3/1/2000, the first working day in the 2nd millennium. On this day, they might not be able to post their entries because the date on that day would be 3/1/00. The program would have problem fixing this in the general ledger.

3. Phone Call:

Suppose you start a phone conversation at 11:55 on 31/12/99 and end at 12:05 on 1/1/00. Again if we are to take absolute values, you would have

spoken for 52034405 minutes (99 * 365 * 24 * 60 +5) instead of 10 minutes. That is disaster, if you are to pay the bill.

4. Research Organizations:

Suppose you are running a time controlled experiment, and the experiment is expected to end at 12:05 on 1/1/2000, else it would blow up the whole laboratory. On that day the lab would be torn to pieces because the date would be 1/1/00.

For the same reason, a rocket billed for firing at the same time would either never be fired or fired before the time.

5. Pharmaceutical:

Pharmaceutical companies would always like to sell off their older drugs before the newer ones, so they would like to sort their drugs by date. However when the bug sets in, drugs manufactured on 3/1/00 would be sold off, only for those manufactured in 1996 to be sold later having expired. Disaster to our health.

It must be mentioned at this point that this problem is not limited to computers alone, it extends to any equipment that is not year 2000 compliants, i.e. equipments that are calendar challenged.

It must be mentioned at this point that this problem is not limited to computers alone, it extends to any equipment that is not year 2000 compliant, i.e. equipments that are calendar challenged.

POSSIBLE SOLUTIONS:

This problem has rightly been described as a time bomb. This means that there is a fixed time for the effects of this bug to start manifestation. We cannot stop the time. Once its time, it kicks off. We are already getting late. We cannot wait any longer than now to start finding ways around the problem.

The first step to the solution of this problem is to create awareness. People should be made aware of its existence and the effect it might have on them. This way they can start taking some precautions. They must be told the truth of the situation. People who are not properly informed have been going about raising false alarm. While some people have been told that their computers would explode in their faces come January 1, 2000, others have been told their's would simply stop working completely.

The truth of the matter is that computers and software produced in the last two to three years do not have this problem, because the programmers have taken the date problem into consideration in their programs. For instance, Windows 95, DOS and other products from Microsoft and other good companies, have taken care of the problem. Even with older PC's, those manufactured around 1980, it is always possible to change their dates without any problem, although when you run programs that are date sensitive, you still run into the millennium bug. However, if what you do with the PC does not involve dates, you have no cause to fear as you computer would continue to work, though with wrong dates. Our problem is with the users of mainframe computers and those with equipments that cannot be reprogrammed because the programs are inbuilt. The problem with the mainframe is that most of the programs used at the time they were manufactured cannot be updated because either their source codes can no longer be found, or because the cost of updating is higher than developing new ones. Unfortunately, many old and big companies and institutions use the mainframe. Newer companies prefer to network their PCs.

Secondly, both PCs and mainframe users need to identify softwares that are likely to be affected and modify them, where possible. One way to identify these programs is to change your system to some date in the year 2000 and try processing some fictitious data and see what the output is. Financial institutions should also try making a post-dated-2000 transaction. If the result looks alright, then they don't have any problem. A better way is use the year 2000 compliant testing program. This program is available on the Internet.

If these tests prove your software is not year 2000 compliant, you must proceed immediately to begin modifying your software, if it was developed in-house and the source code is available, else you need to start thinking or replacing them. That's quite expensive considering that in addition to replacing the programs you have to think of a way to make your old data compatible with the new program.

Anybody affected by this problem should, by the year 2000, begin to process their data manually if the problem is not solved by then.

Other non-computer equipment that are calender challenged are likely to be affected. They must be replaced now if they are suspicious. Factories with automated processes need to watch out for this kind of problem.

Solving the software aspect of this problem might not completely eliminate the bug. The hardware must be able to handle the changes made. It would therefore be necessary to identify the chip that manipulates the date of the system and replace it where possible. Otherwise, doing a partial or total upgrade of the entire system would be a better option, though more expensive.

For computer users, doing proper backup would save you a lot of trouble should you have your data mixed up. With proper backup, you would have something to fall back to when the problem is eventually solved, if ever.

Once the millennium bug problem is solved, the next time we expect a problem

of this sort would be the year 10000, that is, about 8000years from now.

CONCLUSION:

Based on the foregoing discussion, the following conclusions are made:

1. The millennium bug problem exist and must be solved. Solving it requires the contribution of all, because all would be affected, directly or indirectly, when it triggers off at midnight of December 31, 1999.
2. Because there is definite date and time for its kick-off, the right time to solve the problem is now, and in fact, getting late. Time does not, and cannot wait for us.
3. Users of PC's with programs developed in the last 2 -3 years need not fear the bug as it has already been taken care of, but they should assist in creating proper awareness. If they don't, as members of the society, they be would affected somehow.
4. Equipments, other than computers, that are calender challenged must be reprogrammed, if possible, or be replaced.

BIBLIOGAPHY

1. Annonymous: Interview of Senior Colleagues.
2. Okuwoga, O.O. 1997: Millennium bug And The Year 2000. Proc. COAN 6th International Conference.
3. Savage, Senn, 1997: Defusing the "Y2K", Computer Plus Magazine. Issue 6
4. Sweet Michael, 1997: Are Software Bugs Infesting Your Computer? Smart Computing in English, Vol. 8(8).