Panel point of view

Computer Security in the Year 2000[1]

Richard Lefkon
Assistant Professor, New York University
Chair, Year 2000 Committee of AITP SIG-Mainframe
(212) 539-3072

WHY WORRY ABOUT YEAR 2000 SECURITY ?

The problem is, that many legacy applications (custom coded and off-the-shelf) will perform as if the Zero Year is 1900, not 2000, leading to error, malfunction and potentially destructive effects.

Banks, brokers, insurers and healthcare all stand a 2% chance of going out of business that day. Next to this, S&L was puny.

If you like to celebrate New Years Eve - who doesn't? - with hot foods, bright lights, and other energy consumption that will cause your local nuclear power plant to pull those rods at 10:00 PM to meet demand, consider this possible safety instruction: "Replace graphite rods after three hours."
Suppose 1:00 AM never registers on that computer. The authors presentation, "Nuclear Disaster and the Millennium Trojan Horse," treated this and missile mishaps at the 14th NCSC.

BASIC YEAR 2000 SOLUTIONS

You can do this. Once you've chosen among the Seven Methods herein, further technical details are without mystery. Remember that your corporate or agency competitors - and the consulting house team managers - also are learners since at Yearend 1996 fewer than two dozen project managers (the author among them) ever had actually COMPLETED SUCCESSFUL Y2K conversions.

Consider the first two Y2K "Factories," including an Impact Analysis and other steps required 13 years ago when bond trading reports at a large securities firm were seen exchanging profits for losses. The author developed homespun scanner/parsers, which were joined by more sophisticated means the second time. To cut a normally five-year effort by more than two thirds, it was necessary to change and intensify management style, plus win top leadership's backing: (There will always be opposition to your Y2K effort, although not necessarily for the reasons the authors business users had.)

It is imperative to surmount resistance on the part of senior management, to commit resources and funds to allow the I/S department to begin to assess the exposure and to develop explicit plans. Don't permit the mistake of letting Year 2000 compliance be classified as a mere maintenance task. Nebraska raised its sales tax to pay for Y2K, and top management must perceive Y2K as a life-or-death issue. But this, in turn, involves Elisabeth Kubler-Ross's five phases of moving from awareness to acceptance: Denial, anger, bargaining, depression and acceptance. Only at the fifth phase can constructive work begin. It's the IS manager's task to help top management make that trip.

Millennium-proofing your applications will require four basic steps:

Impact Analysis: A thorough inventory of applications and their code, JCL, etc.; automated tools can help find time dependencies, tossable code, and exposure level.
Planning and scheduling: Restructure, redevelop, deploy or retire the systems.
Conversion: Mix among seven alternatives.
Testing and implementation: Use before-and-after year data; tools are available.

The Y2K body of knowledge is old and stable - enough so for NYU to award CEU's in it. Learning here what you need to fulfill your Y2K commitment, doesn't require a Ph. D.

---

[1] This article is adapted from SIG-Mainframes copyrighted reference compendium, Year 2000: Best Practices for Y2K Millennium Computing: Panic in Year Zero. All text is the authors, but in many cases it summarizes articles by the 100 contributors

Be ready to do more testing than you've ever seen before, and adopt approaches that will be new to some:

- Establish a centralized clearinghouse.
- Identify and mitigate risks where the deadline won't be met.
- Investigate for security breaches to occur when systems fail.
- Don't try to solve things by fixing a program at a time.

SEVEN METHODS

There are seven main approaches to the Year 2000 software upgrade task. "Outsource" is not one of them, and if you outsource you'll still need to choose the approach. If you can only list two, you'll have made real decisions about the other five. Deciding not to decide, is a decision.

1. Prune the business.
2. Wait.
3. Replace the application by a purchase or new build.
4. Expand YY year fields to YYYY.
5. "Intelligent" digits, other encoding.
6. Date "Window(s)."
7. Date-shift ("encapsulate") code or data.

Once your orange systems are made Y2K-compliant by one of these methods, if your green ones aren't yet okay, you'll need to "bridge" from one application to another by using intermediate files or other techniques. When you near completion, bridges between conformant applications are removed. Keep them available, though, for times you want to access your archives.

All seven main approaches are easy to understand, especially #2. You barely have time to convert 40% to 60% of your programs fully. Part of your triage approach should be to determine which functions or programs can safely be ignored until next decade because their shortcomings will be cosmetic at worst. For instance, your salespeople won't lynch you for having to scroll past the 12/31 and earlier sales to the 00/04 ones on a "show recent sales first" CICS screen, since this ugliness will only last for days and they'll know their competitors are probably suffering the same inconvenience. Save some cosmetic changes for when there's time.

At the opposite end of "do nothing" are systems which are absolutely critical to the business but have a truly prohibitive Y2K compliance cost. If the company (or line) will become unprofitable due to that outlay, consider shutting it down or selling it. Fast food chains do this frequently with unprofitable stores, and IBM itself has thus far avoided announcing Y2K conformance for the 43xx line of installed mainframes.

Another "pruning" alternative is to sell that business part to a Y2K-ok competitor - but the SEC won't let you foist it on an unsuspecting party.

Third on the list is a full software replacement. This could modernize your business, but you won't be able to regression-test incrementally. A new build is both costly and deadline-risky, and most business-specific software replacements can't help requiring massive customization efforts.

One smart triage using #3 might be to toss your low-datastore Executive Info system, installing a conformant one to be fed by your revamped apps.

Fourth, full YYYY field expansion, is the most appealing and elegant Y2K date fix approach. You regularly make similar changes to keep up with industry wide formats. But there aren't time and resources to expand all to YYYY.

Fifth, "Intelligent digit," is one of three unusual methods. It keeps dates the same length by using the leftmost nibble of the leftmost year byte to hold a century-millennium flag. To use this approach, you have to add various date-encoding and -decoding lines that are anything but obvious. "Seven digit date" is a cross between this method and the one preceding.

Sixth of the seven methods is the so-called century window. Pictorially, this looks just like the kitchen window on the TV series, Honeymooners. The top glass pane represents part of a century of which the YY numbers are above a century "middle" point.

On a two-piece glass window, suppose the middle bar of the window is labeled "1930." All years ending with numbers 30 or above are to be interpreted as "19" plus YY. For instance, "98" is interpreted as "1998,", etc. Expressed as code, IF YY > 30, CC = 19 ELSE CC = 20. Numbers below the bar are future: "20" plus YY. Coding for a century window probably looks familiar, since much of your legacy code already contains date logic for left-affixing "19" or "20."

A so-called "sliding century window" defines the pivot year as being a certain number of years into the past, starting at the current year. Thus, your 100-year window keeps on advancing, preventing shrinkage of the number of years in your future horizon: IF YY > ("now," minus - 50), CC=19.

Last on the list is the quickest - and dirtiest - approach, one which makes no pretense of using the "true" date, as long as the calculations come out right. This date-shift approach downshifts the year by 28. There are seven days to a week and four years to a leap-year cycle; multiplying together and getting 28, you'll find a year shifted this much will have 4th of July and all Sundays in the right place. Taking proper calendar care for new holidays and religious ones, date-related calculations should in theory come out just fine.

Shifting the date requires changing programs (-28/+28 at the start/exit) or changing the data-store to hold (phony) shifted dates, not true ones. But known-incorrect file data raises a possible need to store duplicate data. And so, a code fix is probably the better means to shift dates.

A single canned program segment can repetitively be used to downshift all relevant dates in a given named copybook. The same holds for upshifts. On the whole, "encapsulation" of programs will probably entail the lowest amount of coding among the seven approaches.

Nobody wants to use such an artificial approach, based on false numbers. However, since it is pretty safe and pretty quick, you might want to put a tab labeled "summer 1998" on this page and read it again when time is short and deadlines are close and "Q&D" is less a dirty word.

(A goofy-sounding variation of #7 is to use the Lillian date, storing untrue dates that are the interval since a certain time in history, not untrue dates measured 28 years backward from the present.)

To summarize: There are seven main approaches to cure software that isn't yet ready for the Year 2000. Choosing the best mix for you - even if it's executed by a mediocre workforce - will make you look better than selecting the wrong approach and implementing it through a truly outstanding, top-quality outsourcing firm or in-house team.

This decision is yours to make and cannot safely be outsourced.

QUESTIONS FOR YOU

If you're just starting out, consider the questions which Rep. Steve Horns Government Oversight Technology Subcommittee asked various Federal agencies last spring. Based on their answers, he "gave four As oft of 24 [major departments], and less than a handful of Bs, and the same with Cs, and one heck of a lot of Ds and Fs" to reflect their lack of Y2K preparedness. After the Defense Department received a "C," the U.S. Army announced it would stop routine software enhancements until all Year 2000 problems are fixed. How does your organization score on Horns questions?

> Have you begun a Y2K effort? When started and what steps?
> Risk assessment of vulnerability of programs and applications:
    - If so, show me; if in process, how being performed, when?
> Have you developed a plan?
> Does it have timetables, milestones and performance indicators?
> Do you have a contingency plan? Show me. When invoke it?
> Do you have an inventory of programs, platforms, languages?
    - Users? Lines of code? Lines to be changed?
> Prioritization? OK, which applications are secondary?
> Who is your project manager - Overall? Day-to-day? Tasks?
> What's your org chart? Activities of organization components?
> When were you last given status? Show me the report.
> What resources are/will you devote?
    - Costs by fiscal year, and their components?
    - Can you extract this from existing budget? No: How much?
    - How many person years?

- Estimated cost per line?
- Acquisition plans and their status?
- Proportion of Y2K work to be done in-house vs. outsourced?
- Have you engaged vendors already? Whom and what dollars?

If you intend to get outside help on Y2K, keep these warnings in mind: No tools, just methodology; no methodology, just tools; a "one size fits all" approach; a fully freeform, custom approach; little discussion of testing; no lifecycle; etc.

Ask questions about each software package you use. Is it compliant? If not, when will the vendor make it so, and by what method? What support commitments will the vendor make?

Hold planning and assessment meetings, and follow-up on key vulnerabilities - beginning with noncompliant customers and suppliers. Besides a 10%/80% rule for customer revenue, remember that one tenth your suppliers are critical to your product. To the extent that Y2K failures will shut down businesses financial institutions should watch the 80% of their loan portfolios which consists of loans to businesses.

Finally - and first - survey your applications inventory and platforms. Make sure the app managers and their business counterparts treat your survey seriously. Whether you'll actually use linecount and program count or not, put them in and be suspicious of round numbers. Explicitly ask which of the Seven Methods will be used for each application. Maybe they cant answer correctly yet, but answering will start them thinking.

MORE TESTING THAN YOUVE EVER SEEN

You are almost guaranteed failure if you don't test the software before you start to fix it. Without baseline testing, how do you know whether your wrong results are due to faulty legacy code or your own faulty cures?

Considering the specific re-runs with modified data that you'll have to perform for 1999/12/31, 2000/01/01, 2000/02/29, 2000/03/01 and other key cases, admit to yourself that this time, testing really will take up about two-thirds your effort. The author has even advertised a cash reward of $50,000 to anyone who can disprove this equation computing the maximum number of months (W) that you can wait on an M-month project, before initiating your test environment: $W < M/3 - 2$.
These are the generally accepted logical assumptions:

- It takes up to two months to establish a test environment.
- Everyone agrees testing will cost 55%-75% of Y2K overall.
- Constant overall monthly effort; testing never > 100% of effort.

Basic common sense dictates that testing results after a Y2K code change, must be compared against "baseline" outcomes of the same tests performed before the programs are touched.

Since you'll have to use the exact same user data repeatedly anyway, you
can save time and boredom by using a capture-replay package like these:

Automator, Autotester, CA-Traps, CPR, Carbon Copy, Design Recovery, Hiperstation, Playback, ProTerm, SmartTest, Vermont, etc. Help is even available in composing the test data, via CA-Datamacs, CICS McKinney, File Aid, Grayboxx, Magec, McCabe, Sleuth, STGF/IBM, XDC, etc.

Even better is to search your dusty shelves and discover some excellent test tools you already own. Once you have automated that 55%-75% of Y2K which is testing, you stand a much better chance of reaching all your NON-testing Y2K project goals, too!

Where applicable, non-million-dollar platforms deserve a segregated "time machine" for testing, so that dates can be pushed forward and backward and serious malfunctions produced by the code, without endangering the production machine. In the case of S/390 hardware, you'll also probably want to buy a date simulator to run more than one of the IBM-listed tests at a time. As for an S/390 testing firewall, recently produced inexpensive "pygmy" IBM mainframes provide a means to perform hazardous tests with less risk to the production mainframe than is provided by LPAR and VM Guest alternatives.

By most estimates, the actual code modification to remove Y2K date risk will occupy only about one sixth of your projects labor and costs. Testing will consume four times as much - especially if you don't organize, firewall, and

automate. If you give early attention and resources to Y2K testing, the payback will be four times as great as for most other Year 2000 related activities.

INFORMATION SECURITY EXPOSURE

The last worldwide surprise to InfoSec professionals was the computer virus onslaught at the end of last decade. Here were a limited number of conscious attackers, purposely trying to violate the security of computers at all levels. Generally, they succeeded only at the PC level, and a crop of specialized safety products produced by clever professionals could then and now be purchased by the rest of us as an approximately complete solution.

With Y2K, things are reversed.

Literally millions of "attackers," [the reader included?] have placed billions of Trojan Horses in all kinds of programs and firmware on all platforms. The remediation and future prevention do not require great cleverness, just a prohibitively large collection of ordinary tasks. Also unique is the fact that we must plan for and understand these curative activities, not just charge somebody else's solution to our VISA card and copy a floppy disk to fix everything.