Editor in Chief: Warren Harrison ■ Portland State University ■ warren.harrison@computer.org

The Dangers of End-User Programming

Warren Harrison

The most prominent issue involving software today is security. Virtually every computer user has experienced the results of the variety of worms, viruses, exploits, malware, and denial-of-service attacks across the Internet over the past few months.

Ten years ago, few would have anticipated that a hacker's arrest would make the front page of major newspapers around the world. Today, security tops the concerns not only of the computing industry but also of society in general.

Are software developers to blame?

As I write this, Richard Clarke, who served as chair of the US Critical Infrastructure Board before retiring from government service in 2003, is calling for software vendors to be held responsible for the poor quality and security of their applications. Clarke was quoted as saying at a recent computer security conference, "The reason you have people breaking into your software all over the place is because your software sucks."

In April, the National Cyber Security Partnership (www.cyberpartnership.org) released a report called *Security Across the Software Development Life Cycle*. This report identifies several key issues involved in increasing software security, including

- Enhancing the education and training of present and future developers to put security at the heart of software design and at the foundation of the development process
- Developing, sharing, and skillfully using processes and practices to improve software's quality and security, so that systems are more resilient to attack

Clearly, at no time in the short history of computing has the importance of skilled, professional software developers been so significant. By the very nature of modern security risks, programming mistakes can be devastating not only to the organization using the software but also to other users who are connected to an errant program through networks, shared data, and so on.

Dabblers

Ironically, the news report that carried Clarke's call for software vendors to take more responsibility was followed by an article titled "Debugging for the Masses," which described the EUSES project—End Users Shaping Effective Software (http://eecs.oregonstate.edu/EUSES).

Who are "end user" programmers? They're mechanical engineers, doctors, physicists, teachers, and accountants. They're marketing assistants, receptionists, commodity brokers, and bus drivers. They are, in short, individuals who have taught themselves to program (or perhaps took a course in college from other end-user programmers) in languages such as

0740-7459/04/$20.00 © 2004 IEEE

Visual Basic, PHP, Python, Fortran, and FoxPro. Many of them might even be passable coders. I prefer to call these self-taught coders *dilettantes* (defined as “a dabbler in an art or a field of knowledge”) rather than end-user programmers.

Typically, end-user programmers are employed to perform some other function but write programs to help them in their primary job. However, because programming isn’t their real job, it’s difficult to justify the expense and effort necessary to learn about software development beyond how to put a few programming statements together. Likewise, because end-user programmers usually interact with other programming dilettantes rather than with professional programmers, they seldom know what they don’t know (see my editorial in the last issue, “Clueless—and Oblivious”), and they often have unrealistic ideas of their actual ability.

The EUSES project Web site points out that even though there are 2.75 million “professional” programmers in the US, the number of end-user programmers is expected to exceed 55 million by 2005. The Web site tells us that the project’s goal is to “impact the millions of end-user programmers who, through the use of end-user programming devices, are creating software every day that is, unfortunately, not very dependable.”

You can easily gauge the degree to which this phenomenon has spread by simply browsing the shelves at bookstores. How many *X for Complete Idiots* (where “X” is your favorite programming language) books are out there? I was initially amused by this trend, but recently I’ve become uneasy thinking about where these dabblers are applying their newfound knowledge.

The original end-user programmers were scientists and physicists. After all, the industry moved from assembly language to Fortran in the 1950s because it took too much time for scientists to learn assembly language. When I took my first programming course in 1975, it was with a roomful of engineers and we were learning Fortran. Not surprisingly, I didn’t learn about the concept of systematic testing until 1980 in graduate school.

The next wave of end-user programmers were writers of spreadsheet macros. They were notorious for making mistakes, such as the Florida contractor who used the Lotus 1-2-3 spreadsheet program to prepare a bid in the 1980s. Mistakes in the macro resulted in a bid that was too low to recover costs. After winning the contract, the contractor tried to sue Lotus because of the programming error. While a trained professional developer would have tested the macro to make sure it worked correctly before staking a major bidding decision on the results, the end user omitted this important step.

Spreadsheet programmers were quickly followed by database builders with the introduction of Buttonware’s PC-File, Ashton Tate’s dBase III, and the original FoxPro. These “programmers” usually wrote simple data storage and retrieval applications to keep track of their stamp collections and client lists.

Nevertheless, as long as end-user programmers were limited to developing spreadsheet macros and report-writing software, their impact was generally isolated. However, in the mid 1990s, the Internet boom seemed to make everyone a programmer. Languages such as Perl, Python, and PHP let anyone with access to a Web page tutorial and a text editor write a Web application. I continue to see this trend in the classes I teach at my university. I regularly attract graduate students and faculty from other departments to my classes on developing server-side applications. Most of these individuals will go back to their labs and their departments to create online Web applications.

So, what are they writing?

I found the estimates of the number of end-user programmers sobering. The EUSES Web site continues on to say, “there can be serious consequences for the people whose retirement funds, credit histories, e-business revenues, and even health and safety” rely on software written by end users.

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author’s or firm’s opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society’s Web-based system, Manuscript Central, at http://cs-ieee.manuscriptcentral.com/index.html. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

Can it be true that software manipulating my credit history could have been written by an accountant with no concept of software testing or development processes? How many e-businesses have failed because of lost orders or payments placed through a Web site written by a self-taught Perl or HTML "programmer" who is really a marketing assistant and has never heard of file locking?

And now we're expecting (no, depending on) these folks to write software that not only works correctly but is secure as well?

The effect on all of us

In the 1980s, the impact of dilettantes was fairly isolated. If an end user wiped out his or her database of telephone numbers or appointments, it had minor impact on others. However, we now have systems on the Web that dilettantes built in their spare time while holding down a job in marketing, accounting, hardware repair, or even medicine. They've given little if any thought to systematic testing, maintainability, design, and, yes, security. These systems are available to the entire Internet community—geography and international borders no longer buffer our data from programming mistakes.

The collection and storage of sensitive personal information is so ubiquitous that we've ceased thinking about it: federal ID numbers, credit card numbers, birth dates—even mothers' maiden names (imagine visiting a genealogical site written by a hobbyist). How often do we think about who developed these systems? The process they followed and practices they used?

It's simply unfathomable that we could expect security, as Clarke is demanding, from the vast majority of software applications out there when they're written with little, if any, knowledge of generally accepted good practices such as specifying before coding, systematic testing, and so on. While we all know that using professional programmers doesn't guarantee correctness, security, or maintainability, my money's on the professionals.

What do you think?

I'd like to hear your thoughts on this issue. Is it true that the lack of real understanding about software development by end-user programmers poses a danger to stakeholders associated with mission-critical systems from the standpoints of both correctness and security? Do you have any good anecdotes about end-user programming failures? Let me know. Please write me at warren.harrison@computer.org.

IN OUR NEXT ISSUE

The Business of Software Engineering:

How to Create an Enterprise IT Inventory, and Why

The Social Aspects of Network Effects in Technology Industries

Beyond Black-Box Outsourcing: A Study of 209 Global Projects