One of my colleagues and I enjoy having vigorous discussions which cause those listening to turn pale and back off in fear that we will come to blows.

Actually we’re good friends and just enjoy a good intellectual tussle. Sometimes we’ll switch sides in the middle of the argument for fun.

One of our latest battles practically cleared out the Faculty/Staff dining room in the mess hall at Norwich University last week. The topic was electronic voting systems, and my colleague blew up when I agreed with Dr Rebecca Mercuri that electronic voting systems should produce a paper ballot to be verified by the voter and then dropped into a secured ballot box in case there was a recall.

The details of the argument don’t matter for my purposes today. What fascinated me is his attitude toward the trustworthiness of electronic systems: “That’s ridiculous,” he said. “Surely you should be able to devise a foolproof electronic system impervious to tampering?? Otherwise we’re all in deep trouble, because we’ve been replacing manual systems by electronic systems for years now in all aspects of business. Why should we go to the expense of keeping old manual systems such as ballot boxes and hand recounts – which are vulnerable to abuses anyway – when we can – or ought to be able to – implement completely secure electronic systems?”

This charming confidence in the power of software engineering is undermined by several well-established principles of the field:

- Security is an emergent property\(^1\) (much like usability or performance) and cannot be localized to specific lines of code.

- Testing for security is one of the most difficult kinds of quality assurance procedures known; it is inherently difficult because failures can occur from such a wide range of sources.\(^2\)

- Security failures can come from design errors (e.g., failing to include identification and authentication measures to restrict access to confidential or critical data); programming errors\(^3\) (e.g., failing to implement a security measure because the source code uses the wrong comparison operator in a comparison); run-time errors\(^4\) resulting from poor programming practice (e.g., failing to prevent bounds violations that result in buffer

---

\(^1\) An emergent property in a system is one that cannot be predicted by inspection of the components alone. E.g., volume, reliability, security, safety, maintainability.

\(^2\) Sources of failure include external factors as well as internal problems; e.g., power failures, equipment quality, user error.

\(^3\) Errors made while writing out the instructions in a computer language such as C++, FORTRAN, PASCAL or assembler.

\(^4\) Errors that occur during execution of the program.
overflows and the consequent execution of data as instructions\(^5\); and malicious programming (e.g., logic bombs\(^6\) and back doors\(^7\)).

- Worse, quality assurance is often sloppy, with poorly-trained people who don’t want to be doing the work assigned to the job in spite of their protests. These folks often believe that manual testing (punching data in via a keyboard) is an acceptable method for challenging software (it isn’t\(^8\)); they focus on showing that the software works (instead of trying to show that it doesn’t); they don’t know how to identify the input domains and boundaries for data (and thus fail to test below, at and above boundaries as well as in the middle of input ranges); and they have no systematic plan for ensuring that all possible paths through the code are exercised (thus allowing many ways of using the program to be wholly untested).

- The principles of provably-correct program design have not yet been applied successfully to most of the complex programming systems in the real world. Perhaps some day we will see methods for defining production code as provably secure, but we haven’t gotten there yet.

How ironic that a computer-science geek should thus be in the position of arguing for the involvement of human intelligence in maintaining security. I firmly believe that having independent measures to enforce security is a foundation principle in preventing abuse. Involving skeptical and intelligent people to keep an eye on voting machines is just one example of that principle, and it’s worth the money to prevent our democracy from being hijacked.\(^9\)

\* \* \*

For extensive resources about electronic voting, see Prof. Rebecca Mercuri’s Web site at <http://www.notablesoftware.com/evote.html>.

\* \* \*

M. E. Kabay, PhD, CISSP can be reached by e-mail at <mailto:mkabay@norwich.edu>; Web site at <http://www.mekabay.com/index.htm>.