



IT LEADERSHIP

[TechNewsWorld](#) > [IT Management](#) > [IT Leadership](#) | [Read Next](#)
[Article in IT Leadership](#)

July 15, 2010 02:10:18 PM

Please note that this material is copyright protected. It is illegal to display or reproduce this article without permission for any commercial purpose, including use as marketing or public relations literature. To obtain reprints of this article for authorized use, please call a sales representative at (818) 461-9700 or visit <http://www.ectnews.com/about/reprints/>.

EXPERT ADVICE

Software, Trust and Democracy



By Gwyn Fisher
TechNewsWorld
09/19/08 8:30 AM PT

[Back to Online Version](#)
[E-Mail Article](#)
[Reprints](#)

Developing software for voting machines is a heavy task. When designing code that will be used to measure the decisions of voters in an election and determine new leaders, developers should use an expanded set of tools and techniques to test its security, writes Klocwork CTO

Gwyn Fisher.

▼ advertisement

Find a desktop that uses 50% less power.


HP Compaq 6005 Pro desktops with AMD Phenom and Win 7 use half the energy.

Software is an integral part of everything we do now -- drive a car, make a phone call, turn on the TV, get on an airplane and, yes, exercise your right to vote. Is this the same software that just crashed my new cell phone or sent my credit card number off to a hacker in Eastern Europe? Well, not exactly the same software, but yes, it is software, and it is susceptible to the same sort of risks, with perhaps much more at stake.

The rush for technological solutions (or conveniences) combined with the competitive nature of our society ensures products are brought to market quickly, and e-voting is no different. In this case, first-to-market has also meant major headaches for electoral officials, manufacturers of the e-voting systems, and the voters forced to use them.

The answer chosen by other industries has often been government or industry body regulation, dictating how software must be developed for use within those industries -- for example medical devices, air flight management, payment card transactions, etc.

This regulatory approach is perhaps unrealistic given the decentralized nature of our nation's electoral system, but regulations aside, it's incumbent upon manufacturers of these systems to do the best possible job of designing and writing secure software to ensure the validity of the electoral process. Democracy is not upheld when the system trusts that perfect or uniform

physical security  will be in place or that volunteer electoral workers will always operate the system in its desired state.

A Decentralized System

With such a decentralized electoral system involving rural precincts, thousands of county officials, and e-voting systems that are frequently networked using public infrastructure, the problem isn't trivial.

For many development organizations, this kind of risk management applied to software development is a relatively new idea. E-voting software development organizations need to go through a similar evolution as other software systems used in mission- or safety-critical applications.

What can be done? Development organizations in other safety- or mission-critical software industries have quickly realized there's no silver bullet. When it comes to "building security in" as it relates to software development, there are a range of process improvements that software development teams need to make in order to harden the security and reliability of these systems, which are being increasingly relied upon to deliver consistent, trusted electoral results.

When it comes to validating the security of a software code base (as opposed to designing or testing the system to ensure it functions properly and meets the customer's requirements), there are four areas that every organization should consider implementing at some level: developer training, peer code reviews, security testing and source code analysis.

Developer Training

Regardless of what tools and technologies an organization decides to use, this is where any secure development life cycle needs to begin. The reason is that most developers think of software reliability in terms of whether a particular issue can cause a failure or whether they've developed code in such a way that it will satisfy the design requirements. Both of those considerations need to happen, but developers now need to ask themselves: "Am I writing this software in a way that is making it susceptible to exploit?"

The reality is, most developers probably can't answer that question. Many programming bugs that are considered "security vulnerabilities" can often seem quite innocuous to the untrained eye, so it's critical that developers are not only provided with the proper tools to write code securely, but also equipped with the knowledge to remediate any issues. This type of ongoing education will go a long way to helping developers understand whether their coding practices are defensive enough to uphold the validity of our country's electoral process.

Peer Code Review

Probably the most maligned (often for good reason!) form of software validation is the dreaded peer code review, which basically means sitting in a room with colleagues and senior designers and staring at the code that's been written to ensure security. Many organizations

outsource this task, which is a very legitimate approach, but it's good practice to have some level of peer code review take place in-house on a regularly scheduled basis.

The challenge with an over-reliance on this technique is that it's not scalable across large code bases and is obviously prone to some level of human error. Given that, many organizations focus the review activities on specific software components that are particularly security-sensitive or complex. This is often an effective approach and allows an augmentation of the more automated techniques available.

Testing for Security

The most popular -- and in some ways the least effective -- approach to addressing this problem is to place the burden on the QA (quality assurance) or pre-QA testing functions in an organization -- the so-called "testing security in" approach.

Tools such as penetration test and fuzz test are some of the most popular technologies used by testers to provide some comfort level that the product is secure. While there are variations in particular test technologies, in general, they all apply a similar approach in principle. That is, they bombard the application with a library of well-known (and, if the tool provider is any good at what they do, not-so-well-known) attack patterns to see if it will allow unintended access to what should be protected subsystems. The attack patterns are supplied by the tool vendor and are updated regularly, so this is a very valuable technique for "covering the waterfront" in terms of common attack patterns.

Of course, what these tools validate is not that the system under test has no exploitable vulnerabilities, but rather that given the available catalog of known attack patterns, none of them yield to overt exploit. This "closed ended" test approach is therefore valuable as a baseline, but it should never be confused with providing a reasonable assessment of holistic system security.

The real challenge with electoral systems is the magnitude of what's at stake, and therefore the lengths to which malicious attackers will exert themselves to gain an undemocratic advantage. If your answer to this challenge is to simply punt the problem downstream to QA, you have failed the voter before you even begin.

The reality of a complex and vibrant software system of this type -- or indeed any system of reasonable size and complexity -- is that it will generate a virtually infinite number of software runtime paths, especially when error handling routines are brought into the equation. Using a "test security in" approach exclusively is always going to fail in the face of such a combinatorial explosion.

As stated above, while security testing provides a solid baseline for confidence, confusing this confidence with "it's secure" is false. Security testing always needs to be augmented with additional techniques targeting upstream aspects of the development process in order to achieve decent levels of risk mitigation.

Source Code Analysis Tools

Clearly, any proper risk mitigation strategy needs to implement the broadest set of tools and processes that are realistically possible. It's perhaps obvious, although not always well understood, that regardless of what other security precautions are taken, if the source code itself contains programming errors that are susceptible to exploit, the system can never be secure. As mentioned, "early and often" peer code review is one approach to mitigating areas of risk within the source code, but given the volume of code in systems today, automation is obviously called for.

Source code analysis (SCA) technology is an automated approach to this problem and is designed to locate and describe areas of weakness in software source code. The technology is distinct from more traditional dynamic analysis techniques such as fuzz or penetration tests, since the work is performed at build time using only the source code of the program or module in question. Good SCA tools can deliver accurate analysis directly to the developer, allowing them to identify and remediate coding vulnerabilities as they're developing the software. When these tools are put directly in the hands of developers, fewer coding vulnerabilities will make it into the code stream, leading to more secure code in addition to greater efficiencies and focus during peer code reviews and QA security testing.

Improvements in polling system technology are long overdue. Many states and counties are still using 19th century technology, which, while generally effective, is susceptible to error and not scalable given the population growth since these systems were first introduced. So the opportunity for new electoral systems technologies is exciting and welcome. Given that software drives many of these systems, proper validation needs to be implemented by the manufacturers. The required changes in tools and processes are widely available and used in other mission-critical software industries. [ECT](#)

Gwyn Fisher is chief technology officer at [Klocwork](#), a provider of source code analysis tools.

Next Article in IT Leadership

[Dark Data: What You Can't See Can Hurt You](#)
September 18, 2008



Many physicists theorize that the majority of the universe is made up of dark energy and dark matter -- they can't see it, but indirect evidence repeatedly suggests it's there. Similarly, enterprise networks are often chock full of what columnist Ed Moyle calls "dark data" -- you know it's there, but you can't be sure what it is or what it does, and that's not good.

Copyright 1998-2010 ECT News Network, [Terms of Service](#) | [Privacy Policy](#) | [How To Advertise](#)
Inc. All Rights Reserved.