Trust, but Verify

Michael Lesk | Rutgers University

Most software is sold in binary form, and purchasers must rely on the vendor’s promise that the software is both reliable and secure. We might prefer software sold as source code so we can check it, but then the vendor has to trust purchasers to keep the product confidential. Open source is a model with no confidentiality, leaving the vendor only able to sell support. Are there economic models to enable a software industry that offers assurances with less trust? One answer is third-party testing, and perhaps we should use it more often.

Testing is often separated into “black box” and “white box” (see Figure 1). In black-box testing, you can’t inspect source code; you can only run the program. This is useful, for example, if you want data on how long the program takes using different input. White-box testing can include code inspection, which is necessary for security. Both software checkers and human eyes examine the code to determine whether important precautions, such as input validation, have been taken. Testers also look for code weaknesses—both specific dangers and general coding errors.

Elections, Gambling, and Hospitals
Suppose you run elections and want to buy voting machine software. You can do so from vendors such as Diebold, but you don’t get the source code. How do you know that the votes are counted correctly? For years, researchers such as Avi Rubin have been frustrated that the voting machine industry won’t let you see its code. The cynical or paranoid will note that Wally O’Dell, the president of Diebold Election Systems in 2004, promised that he would try to deliver Ohio for George W. Bush.1

Even if Diebold isn’t attempting to change election results, how do we know that its system is resistant to malicious outsiders? In 2010, the District of Columbia connected its voting software to the Internet and invited trial attacks. Some folks in Ann Arbor broke into the software and arranged it so that when someone cast a vote, the machine would play the University of Michigan fight song.2 The District understandably decided against that particular voting technology.

In contrast, suppose you want to set up a video poker terminal in a Las Vegas casino. Remember the scene in Casablanca where Herbert Evans’ character asks whether the roulette wheel in Rick’s back room is honest, and S.Z. Sakall’s character replies, “as honest as the day is long”? Well, if you want to set up a poker machine in Nevada, you’re required to send your source code to the Nevada Gaming Commission, which evaluates the code to ensure it conforms to your promises and its expectations about payout rates and fairness. The Commission has the right to pull the chips out of your machines on the casino floor and check that the code you’re running is the code it has on deposit. We seem to care more about the honesty of our casinos than our elections.

There are many kinds of software you might care about. For example, you’d probably like to believe that electronic health records are both reliable and secure—and not just to preserve patient privacy while delivering the right health information needed for treatment. For instance, if attackers can break into an e-prescribing system and write prescriptions for expensive chemotherapeutic drugs, they can sell the drugs on the black market. Another threat was realized two years ago: hackers broke into an Illinois health clinic’s computers, encrypted all patient records, and demanded money for the password.3

Independent Validation
How should a hospital judge whether its software is secure? In 1994, Denmark created MedCom, an independent nonprofit organization that validates software for health records and communication. For example, suppose 10 different vendors offer to sell communication systems to physicians. MedCom would test these new offerings; for communications, this takes one week.4 In total, MedCom has 15 employees, including testers and validators of other health-related software.

11/13/14 2:58 PM
In the US, electronic health record system vendors react with horror at the idea of submitting their code for outside analysis, perhaps by the US Food and Drug Administration; they assume that the approval process would be extremely long or expensive. I have the reverse fear: although the Danish Data Protection Agency backs MedCom, the Danish process is so fast that I worry it might be too perfunctory.

In the US, the National Institute of Standards and Technology (NIST) helps with software assurance. NIST’s Software Assurance Metrics and Tool Evaluation project develops tools for security analysis, and its Security Content Automation Protocol program validates systems that conform to its security guidelines. NIST, other government agencies, and contractors compile lists of known vulnerabilities and best practices for security; Carnegie Mellon University’s computer emergency response team is perhaps the most prominent. Even unexpected organizations, such as the Department of Energy’s Fermi- lab, publish instructions on reviewing code for security.

Commercial Validation
Commercial companies also perform this service. For example, Diebold employs SAIC to inspect its code and validate that votes will be counted correctly. The banking folks have the Payment Card Industry Security Standards Council, which maintains a list of qualified security assessors who test conformance to its security rules. More broadly, companies such as Praetorian, Veracode, Foundstone (part of MacAfee), Software Secured, and Coverity inspect code with the aid of automated tools. A serious inspection is not cheap. Veracode inspections start at US$100,000.6 Praetorian estimates that a security consultant can read 2,000 to 2,500 lines of code per day; think about that rate applied to a big program. In practice, manual inspection should focus on critical parts of the code; thus, every contract will be site specific.

Across the software industry, the cost of pulling bugs from a program is generally comparable to the cost of writing it in the first place. Thus, half the budget for security-critical coding should go to security assurance and testing. If software maintenance isn’t expensive, it’s more likely that nobody is using the code than that there aren’t bugs in it. However, everybody is in agreement that efforts spent to write the original code well are much more cost-effective than fixing it later. Even before adherence to good coding standards, compliance with security guidelines, and unit testing, analysis of the requirements for security risks and examination of all interfaces to other systems are valuable.

Although both government and private groups perform software verification, few ads for software boast of being inspected for security, and neither the government groups nor the private groups I mentioned here are well-known at the consumer level. You can buy “cyberliability” insurance, but there’s no well-known equivalent of the Insurance Institute for Highway Safety, which rates cars. Historically, many kinds of insurance have come with inspections and rules. Benjamin Lee Whorf, one of the greatest American linguists, had a day job with the Hartford Fire Insurance Company advising customers on how to reduce their risk. As an example of what insurance companies did then, Whorf advised a customer that employees smoking near full gasoline drums was less dangerous than doing so near empty drums, because the empty drums were full of vapor. Vapor is more explosive than liquid gasoline, and those “empty” drums were a greater risk. I see no insurance company today providing the same high-level advice about software safety.

Figure 1. Software testing. (a) In “black-box” testing, you must run the program as given. (b) In white-box testing, source code can be inspected, which is essential for security validation. This particular cat is good at spotting bugs, but unfortunately only the literal, not the metaphorical, kind.
In the July/August 2012 Security & Privacy Economics department, “Georgia on My Mind” (vol. 10, no. 4, pp. 88–90; http://doi.ieeecomputersociety.org/10.1109/MSP2012.97), Michael Lesk discussed the May 2012 district court decision in a 2009 copyright lawsuit of several major publishers against Georgia State University. On 17 October 2014, the 11th Circuit Court of Appeals overturned that decision, and the district court will have to rule again. Expect another little box here in two more years.

—Eds.

Open Source

Because anyone can examine its source code, open source software offers the possibility of crowdsourced debugging. Indeed, Coverity reports that it finds fewer bugs in open source than in proprietary software.7 Yet, bugs do appear in open source software. A prominent recent bug was OpenSSL’s Heartbleed, a coding error made in late 2011, distributed in March 2012, and found in April 2014. This bug, a simple duplication of a line of code, wasn’t detected by Coverity’s static analyzer but by manual examination of code at Google and Codenomicon (a Finnish security company). It’s unclear whether any “bad guys” knew about the bug before it was publicized; it was announced on 3 April, and break-ins occurred on 8 April in at least two places.

Heartbleed demonstrates that open source isn’t a panacea for security issues. Ian Levy of the UK’s Government Communications Headquarters feels that neither proprietary nor open source software is automatically safer or riskier: “Many eyes give you many eyelashes, and not a lot else.”8 Much open source software is essentially anonymous, and as Levy noted, a user has no easy way to know whether developers took security seriously, followed good design principles, or took other precautions. In contrast, you might be able to ask a commercial supplier these questions.

Recommendations

The real problem isn’t whether we use government or private code verification. The real problem is that for too many products, we don’t do either. Few people know what’s in any Android apps they download, think before giving approval to an app for any permissions it asks for, or read the end user license agreement for any ordinary piece of consumer software. We have no equivalent of an Underwriters Laboratory for software, and we need one.

The code in SSL that was vulnerable to Heartbleed wasn’t written in a way to be easy to inspect, and therefore the weakness wasn’t spotted earlier. The duplicated line of code was a “goto,” which has been discouraged for use in control flow since the 1970s. We really don’t want security through obscurity; good programming principles should be followed in all important projects, just as we expect in bridge and building engineering.

We need principles and a validation industry. Yes, a product’s raw code will cost more, perhaps 50 or 100 percent more, but we need to see such software advertised as “verified by...,” and we need purchasers to recognize this validation. We shouldn’t just trust software suppliers, whether commercial or volunteer, without a reason to believe their code is secure.

References

1. M. Warner, “Machine Politics in the Digital Age,” 9 Nov. 2003, New York Times, p. Business-1.
2. N. Casal Moore, “Researchers Hack into DC Voting System Test Bed, Leave Fight Song Signature,” Michigan Eng., 7 Oct. 2010; www.wengin.umich.edu/college/about/news/stories/2010/october/researchers-hack-into-dc-voting-system-test-bed-leave-fight-song-signature.
3. J. Robertson, “Hackers Encrypt Health Records and Hold Data for Ransom,” Bloomberg News, 10 Aug. 2012.
4. D. Protti and I. Johansen, “Widespread Adoption of Information Technology in Primary Care Physicians in Denmark: A Case Study,” Issues in Int’l Health Policy, Commonwealth Fund, vol. 80, no. 1379, Mar. 2010.
5. G. Garzoglio, “A Code Inspection Process for Security Reviews,” FERMILAB-PUB-09-234-CD, Fermi National Accelerator Lab, 2009; http://lss.fnal.gov/archive/2009/pub/fermilab-pub-09-234-cd.pdf.
6. “Veracode Offers On-Demand Security as a Service,” Software Magazine, Feb. 2007; www.software.com/content/ContentCT.aspx?P=2849.
7. J.E. Dunn, “Open Source Trounces Proprietary Software for Code Defects, Coverity Analysis Finds,” Techworld, 16 Apr. 2014.
8. N. Heath, “Six Open Source Security Myths Debunked—and Eight Real Challenges to Consider,” ZDNet, 23 Apr. 2013.

Michael Lesk is a professor at Rutgers University. Contact him at lesk@acm.org.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

Got an idea for a future article? Email editors Michael Lesk (lesk@acm.org) and Jeffrey MacKie-Mason (jmm@umich.edu).