Does Open Source Improve System Security?

Brian Witten, Carl Landwehr, and Michael Caloyannides,
DARPA and Mitretek Systems

An attacker could examine public source code to find flaws in a system. So, is source code access a net gain or loss for security? The authors consider this question from several perspectives and tentatively conclude that having source code available should work in favor of system security.

The current climate of functionality and performance-driven markets has created enormous code bases, which have helped drive growth in the US gross domestic product in recent years. However, these code bases have also created an information infrastructure whose vulnerabilities are so striking as to endanger national and economic security.[1] Distributed denial of service attacks have demonstrated that such vulnerabilities can degrade the Internet's aggregate performance,[2] and recurrent virus outbreaks have inflicted substantial repair and recovery costs on businesses worldwide.[3]

There is no guarantee that making a system's source code publicly available will, by itself, improve that system's security. Quantifiable arguments could help us make that decision, but we don't have good measures for how secure systems are in the first place.[4] There is substantial literature debating the merits of open source, but much of it is based on examining a few examples anecdotally. Hard data and rigorous analysis are scarce.

Open review of algorithms, particularly cryptographic algorithms, has long been advocated,[5] and its benefits are hardly debatable.[6] But ultimately, Rijndael, the selected Advanced Encryption Standard algorithm, will operate as software and hardware. If we can openly review the algorithm but not the software, there is ample room for doubt as to whether a particular implementation is trustworthy. There are many definitions and degrees of open source-ness.[7]

For this discussion, we simplify the question to whether making source code freely available for security review, and potentially repair, is more likely to improve or degrade system security. Revealing a system's source code is not just a technical act. It has economic and other effects that we must consider.

Defender's view: Closed doors

In restricting the release of source code, producers require consumers not only to blindly accept that code but also to trust the compiler employed to create it. Consumers thereby forfeit many abilities to enhance the final executable's security. Ken Thompson demonstrated how to create a

0740-7459/01/$10.00 © 2001 IEEE

compiler whose subverted code you cannot find by reviewing its source code.[8] With such a compiler, even a well-intentioned developer would unwittingly produce flawed executables.

Having the source code available creates opportunities for defending against both accidental and malicious faults. Although an open source system user is vulnerable to the tools he or she uses to generate the executable, the choice of tools is the user's. If caution dictates, he or she can introduce variability by, for example, using different compilers to generate the executable. Moreover, automated tools can scan source code to identify potentially dangerous constructions,[9] which can then be modified manually or, in some cases, automatically. Compilers can also help strengthen a system's security without modifying source code. For example, the Immunix Stackguard compiler adds "canaries" that defeat many buffer overflow attacks.[10] This technique adds the canaries into the executable code without modifying the source code, but it requires access to the source code.

Similarly, compilers that randomize their output in various ways[11] can defeat other attacks that exploit specific, otherwise predictable details of compiled programs. These approaches can remove vulnerabilities at a much lower cost than manual source code modification, but they cannot work unless the source code is available.

Without access to the source, defenders cannot invest in having humans review their source code, which is still crucial to investigating specific hypotheses about back doors, race conditions, and other subtle flaws that continue to plague our systems.

Java Obfuscation/ Decompilation URLs

C.Wang, *A Security Architecture for Survivability Mechanisms*, PhD dissertation, Univ. of Virginia, Dept. of Computer Science, Oct. 2000. Includes survey (pp. 178–180) of code obfuscation techniques as well as original research: ftp://ftpcs.virginia.edu/pub/dissertations/2001-01.pdf.

Condensity, a commercial product for obfuscating Java byte code: www.condensity.com/support/whitepaper.html.

List of Java decompilers: www.meurrens.org/ip-Links/Java/CodeEngineering.

Attacker's view: Open opportunities?

Providing public access to source code means the attacker has access to it as well. Relative to cryptographic algorithms, the source code for, say, an operating system is likely to be large and complex and will almost certainly contain some exploitable flaws. On the other hand, the attacker will in any case have access to the object code and, with suitable resources, can probably reconstruct any portion of it. Or, the attacker could obtain a copy of the source illicitly, if he or she is well funded or has appropriate connections to the developer.

Closed source is a mechanism of commerce, not security, and it relies on law enforcement to prevent abuses. Compilers are not designed to provide cryptographic protection for source code, as evidenced by the market in tools that try to obfuscate Java bytecode as well as decompilers to defeat the obfuscators (see the sidebar). Thus the difference between open and closed source models might not be so great as imagined, serving primarily to deter those who are basically law-abiding or whose resources are limited. A second factor, considered in more detail later, is whether open or closed source systems are more likely to have known but unpatched vulnerabilities at any particular time. The recent Digital Millennium Copyright Act and UCITA legislation seem designed to discourage law-abiding citizens from reconstructing source code to improve its security but are unlikely to deter those with baser motives.

Although we have primarily limited our concern to source code that is available for open review (or not), in some open source environments the attacker might have the opportunity to corrupt the code base by submitting subtly sabotaged programs to be included in the public system. Of course, we do not lack for examples in which the authorized employees of prominent software developers have inserted trapdoors and Easter eggs in systems without management's apparent knowledge or consent.

Economics: Who pays the piper?

According to the old adage, he who pays the piper calls the tune. But, in the market

for closed source operating system software, the payers are many and small, and the pipers are few and large—the payers dance to the piper's tune.

Closed source preserves the producer's economic interest and can allow recovery of development costs through product sales. Although certain licensing arrangements afford source release in a protected manner, it is commonly believed that restricting it helps preserve the intellectual property contained in an executable.

Once the initial version of the product has saturated its market, the producer's interest tends to shift to generating upgrades. Because software isn't consumed or worn out by repeated use, the basis for future sales depends on producing new releases. Security is difficult to market in this process because, although features are visible, security functions tend to be invisible during normal operations and only visible when security trouble occurs.

Anyone other than the producer who wants to invest in improving a closed source product's security will have a hard time doing so without access to the source code. Some might be willing to invest in the producer and trust that their investments will go toward the product's security. Justifying such investments is hard because the benefits accrue to the single producer, whose incentive is to keep producing upgrades, which in turn tend to require renewed investment in security.

The economics of the open source model seems mystical—through the efforts of volunteers, a stable product appears and grows. How do these people eat? Yet, this market's vitality is undeniable. Part of the answer is that the expertise individuals and companies develop in the open source product is salable, even if the product itself is freely available. Another part might be that the security market is roughly evenly split between products and services, which has created a community of security professionals capable of assisting the development and maintenance of open source products.[12]

Metrics and models: Describing the elephant

It would help to have empirical support for this discussion to a greater extent than we do. To decide objectively whether open source will improve system security, we need to at least rank order the security of different systems (or of the same system, under open and closed source conditions). But, security is a little like the elephant of Indian lore—the appropriateness of a measure depends on the viewer's perspective. What kinds of measures would be appropriate?

A source of reliability—if not security—metrics are the "fuzz" tests of Unix utilities and services performed at the University of Wisconsin first in 1990 and again in 1995. The testers concluded that the GNU and Linux utilities had significantly lower failure rates than did the commercial Unix systems.[13]

Development of proper metrics for system security is a vast topic—often addressed but rarely with satisfying results—but it exceeds this brief article's scope. Nevertheless, SecurityPortal has proposed a simple and plausible metric relevant to the case at hand. Arguably, systems are most vulnerable during the period after a security flaw becomes known to attackers and before the flaw is removed (such as by distributing a patch). So, one measure of system security might be this interval's average length. Data measuring this interval are hard to come by, but an effort to characterize an open source system (Red Hat Linux) and two closed source systems (Sun and Microsoft) based on published security advisories is available at www.securityportal.com.[14]

The results on SecurityPortal's Web site show an average number of days of vulnerability for Red Hat Linux as 11.2 with standard deviation 17.5, based on 31 advisories. For Microsoft, results show an average of 16.1 days with a standard deviation of 27.7, based on 61 advisories. Sun shows an average of almost 90 days, but only eight advisories were published during the year, making statistical inference questionable.

Moreover, SecurityPortal concluded,

> *Red Hat could have … cut their turn around time in half, had they only been more attentive to their own community. There were instances when software had already been patched by the author, but Red Hat was slow in creating RPM distribution files and issuing advisories.*[14]

Open source advocates should not be surprised to find that the proprietary, central-

Security is a little like the elephant of Indian lore—the appropriateness of a measure depends on the viewer's perspective. What kind of measures would be appropriate?

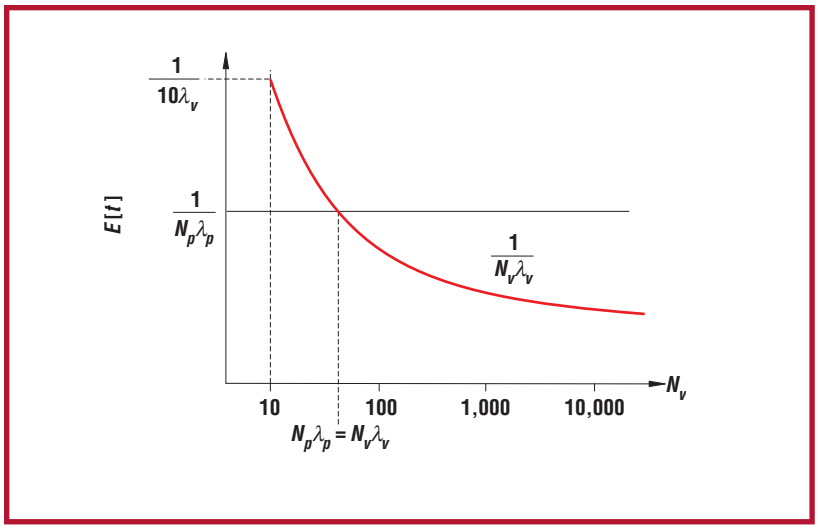

Figure 1. Expected time to find next security flaw ($E[t]$) versus number of volunteer reviewers in open source community (N_v).

ized distribution system for the open source software seems to double the delay in fielding patches.

Many caveats go with this kind of data, including

- not all flaws are equally bad;
- finding many flaws but repairing them quickly might not be better than finding few and taking longer to fix them;
- normalizing results from different operating systems might affect results; and
- the effectiveness of putative fixes might differ.

Nevertheless, making such measurements available on a routine basis could go far in helping consumers make better-informed purchasing decisions.

Another approach is to look at a simplified statistical model of how the open and closed source review processes seem to work. In the closed source case, suppose there is a relatively fixed size set of paid source code reviewers. They are presumably knowledgeable and motivated. In the open source case, there is a more heterogeneous group of volunteers. We propose, with some justification from previous models[15] and experiments,[16] to use a Poisson process to describe the rate at which each individual finds security flaws. To account for presumed differences in motivation and expertise, we choose λ_p to represent the rate at which a paid reviewer finds flaws and λ_v to characterize the rate for volunteer reviewers. If the number of paid reviewers is N_p, then the expected time for the paid group to find the next flaw is simply $1/(N_p\ \lambda_p)$, and the rate for the volunteer group is similarly $1/(N_v\ \lambda_v)$. This model does not address the different coefficients of efficiency for group collaborations. Claims of advantages in emergent behavior weigh against claims of advantages of central control, however, and we have not yet found any hard data on which to base these coefficients. Moreover, this model doesn't adjust for the rate decreases that might be associated with depletion of vulnerabilities. However, this model should be adequate for at least beginning to provide structure to some of the arguments on both sides. Figure 1 gives an idea of the potential effect of a larger group of security reviewers, even though they might be, on average, less expert than paid reviewers. This simple analysis highlights that past the point where $N_v\ \lambda_v = N_p\ \lambda_p$, the open source approach detects errors more quickly. There's a similar effect when groups of students review security requirements.[17]

Today, it seems that all software has holes. We can draw four additional conclusions from this discussion. First, access to source code lets users improve system security—if they have the capability and resources to do so. Second, limited tests indicate that for some cases, open source life cycles produce systems that are less vulnerable to nonmalicious faults. Third, a survey of three operating systems indicates that one open source operating system experienced less exposure in the form of known but unpatched vulnerabilities over a 12-month period than was experienced by either of two proprietary counterparts. Last, closed and proprietary system development models face disincentives toward fielding and supporting more secure systems as long as less secure systems are more profitable. Notwithstanding these conclusions, arguments in this important matter are in their formative stages and in dire need of metrics that can reflect security delivered to the customer.

More caveats are also necessary. There is little evidence that people (in particular, experienced security experts) review source code for the fun of it. Opening the source code creates the opportunity for individuals

or groups to understand how it works and what security it provides, but it cannot guarantee that such reviews will occur. Some vendors of closed source products have regulated internal processes for reviewing and testing software. Many will make their source code available to qualified reviewers under nondisclosure agreements. There is, in any case, no guarantee that human review will find any particular security flaw in a system of millions of lines of code—especially not if there is a serious effort to hide that flaw. Review will always be most effective on small, well-structured pieces of code.

Still, closed development models force consumers to trust the source code and review process, the intentions and capabilities of developers to build safe systems, and the developer's compiler. Such models also promise to deliver maintenance in a timely and effective manner, but consumers must also forfeit opportunities for improving the security of their systems.

Acknowledgments

This article has benefited from the comments of anonymous referees. In addition, we thank Rick Murphy of Mitretek Systems and Steve Lipner and Sekar Chandersekaran of Microsoft, who provided helpful reviews of a draft of this article, but who do not necessarily concur with its conclusions!

Disclaimer

The views expressed in this article are those of the authors and not the Department of Defense or Mitretek Systems.

References

1. US Government Working Group on Electronic Commerce, *First Annual Report*, US Dept. of Commerce, Washington, DC, Nov. 1988, pp. 1–2, www.doc.gov/ecommerce/E-comm.pdf (current 11 July 2001).
2. L. Garber, "Denial of Service Attacks Rip the Internet," *Computer*, vol. 33, no. 4, Apr. 2000, pp. 12–17.
3. A.C. Lear, "Love Hurts: New E-Mail Worm Afflicts Millions," *Computer*, vol. 33, no. 6, June 2000, pp. 22–24.
4. Computer Science and Telecommunications Board, *Trust in Cyberspace*, National Research Council, Washington, D.C., 1999, pp. 185, 189.
5. A. Kerkhoffs, "La Cryptographie Militaire," *J. des Sciences Militaires*, vol. 9, Jan. 1883, pp. 5–38.
6. D. Wagner, B. Schneier, and J. Kelsey, "Cryptanalysis of the Cellular Message Encryption Algorithm," *Counterpane Systems*, Mar. 1997, www.counterpane.com/cmea.pdf (current 11 July 2001).
7. V. Valloppillil, *Open Source Software: A (New?) Development Methodology*, Microsoft Corp., Redmond, Wash., 11 Aug. 1998; www.opensource.org/halloween/halloween1.html (current 11 July 2001).
8. K. Thompson, "Reflections on Trusting Trust," *Comm. ACM*, vol. 27, no. 8, Aug. 1984, pp. 761–763.
9. J. Viega et al., *A Static Vulnerability Scanner for C and C++ Code*, Cigital, Dulles, Va., 2000; www.cigital.com/papers/download/its4.pdf (current 11 July 2001).
10. C. Cowan, "Automatic Detection and Prevention of Buffer-Overflow Attacks," *Proc. 7th Usenix Security Symp.*, Usenix, San Diego, 1998, pp. 63–78.
11. S. Forrest, A. Somayaji, and D. Ackley, "Building Diverse Computer Systems," *Proc. HotOS VI*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 67–72.
12. C.J. Wilson, "Graphiti," *Red Herring*, no. 77, Apr. 2000, pp. 68–70.
13. B.P. Miller et al., *Fuzz Revisited: A Reexamination of the Reliability of Unix Utilities and Services*, tech. report, Computer Science Dept., Univ. of Wisconsin, Madison, 1995, www.cs.wisc.edu/~bart/fuzz (current 11 July 2001).
14. J. Reavis, "Linux vs. Microsoft: Who Solves Security Problems Faster?" SecurityPortal, 17 Jan. 2000, www.securityportal.com/cover/coverstory20000117.html (current 11 July 2001).
15. B. Littlewood et al., "Towards Operational Measures of Computer Security," *J. Computer Security*, vol. 2, no. 3, Apr. 1993, pp. 211–229.
16. E. Jonsson and T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior," *IEEE Trans. Software Eng.*, vol. SE-23, Apr. 1997, pp. 235–245.
17. R.J. Anderson, "How to Cheat at the Lottery," *Proc. 15th Ann. Computer Security Applications Conf.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. xix–xxvii; www.cl.cam.ac.uk/~rja14/lottery/lottery.html (current 11 July 2001).

Review will always be most effective on small, well-structured pieces of code.

About the Authors

Brian Witten is a program manager at DARPA, where he has directed several programs including the Information Assurance, Autonomic Information Assurance, Partners in Experimentation, Survivable Wired, and Wireless Infrastructure for the Military programs. He received his BS in electrical and computer engineering from the University of Colorado. He is a member of the IEEE. Contact him at DARPA/ATOO, 3701 North Fairfax Dr., Arlington, VA 22203-1714; bwitten@darpa.mil.

Carl Landwehr is a senior fellow at Mitretek Systems, working closely with the director of Mitretek's Information Security and Privacy Center. He provides senior technical support to several program managers within DARPA's Information Assurance and Survivability and Third Generation Security Programs and assists the Federal Infosec Research Council. He received his BS from Yale University and an MS and PhD in computer and communication sciences from the University of Michigan. He is a senior member of the IEEE and has served as an associate editor of *IEEE Transactions on Software Engineering*. Contact him at Mitretek Systems, MS Z285, 7525 Colshire Dr., McLean, VA 22102; Carl.Landwehr@mitretek.org.

Michael Caloyannides is a senior fellow at Mitretek Systems concentrating on computer forensics, encryption, and privacy technical problems. His interests include information security and telecommunications. He received a BSc and an MSc in electrical engineering and a PhD in electrical engineering, applied mathematics, and philosophy, all from Caltech. He holds one US patent on high-speed modems and is a senior member of the IEEE. Contact him at Mitretek Systems, MS Z285, 7525 Colshire Dr., McLean, VA 22102; micky@IEEE.org.