Open Source in Security: Visiting the Bizarre*

Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853
fbs@cs.cornell.edu

Abstract

Although open-source software development has virtues, there is reason to believe that the approach would not have a significant effect on the security of today's systems. The lion's share of vulnerabilities caused by software bugs is easily dealt with by means other than source code inspections. And the tenets of open-source development are inhospitable to business models whose success depends on promoting secure systems.

Feature Enhancement Dominates. A principle tenet of open-source development is that source code be available for review and modification. For systems having a large developer base (infrastructure software might, specialized applications won't), this means that many are inspecting and improving the code. But one must be careful to distinguish between what is possible and what is probable.

There is no reason to believe that the many eyes inspecting (open) source code would be successful in identifying bugs that allow system security to be compromised. In fact, one could argue that such bugs would likely elude open-source developers. Developer-perceived improvements are what drives code-base evolution for open-source systems. Open source developers—as opposed to hackers—have no more incentive to invest time and energy in code audits to uncover security vulnerabilities than closed-source developers do.

*Supported in part by ARPA/RADC grant F30602-96-1-0317, AFOSR grant F49620-94-1-0198, Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-99-1-0533, National Science Foundation Grant 9703470, and a grant from Intel Corporation. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon.

Closed-source software development is driven by market-entry time and features, because the industry believes that's what the market wants. Testing and other activities that increase quality but delay deployment are thus eschewed by closed-source software developers. And the quality of their systems suffers for it. But a change in the market could reverse that. Such a change might result from adverse publicity or from legislation that makes developers liable for flaws in their software.

With open-source software, the same change in the market would affect only the value-added service providers since customers, wanting to have someone to hold accountable for claims about assurance, would obtain such added value in this way. These service providers, however, could not justify significant investments aimed at raising assurance. First, any bugs they identify and repair would, as proscribed by the tenets of open-source development, be reflected in everyone's code base—the return on investment in this activity is thus diluted. Second, today's techniques for establishing assurance are not incremental. A constantly evolving code base, as is likely because of the large number of developers in open-source development, would thus require continuing, significant investments.

Grease the Squeaky Wheel. Systems today are compromised not only because hackers exploit bugs but for prosaic reasons: initial configurations are shipped with defenses disabled, administrators misconfigure their systems, and users find it inconvenient to set protections for their files or encrypt their data. Some of these vulnerabilities would be addressed with easier-to-use security functionality while others require a new culture of caution and care amongst users. None has anything to do with whether a system is open-source or closed-source. Thus, if bugs in the code base are not today the dominant avenue of attack, then embracing open-source software is applying grease to the wrong wheel.

Even if we restrict attention to vulnerabilities caused by software bugs, which open-source facilitated inspections

supposedly target, reviewing the CERT advisories and other compendia of vulnerabilities finds a single class of bug predominant: buffer over-runs. Use of programming languages that enforce weak forms of type correctness would prevent such bugs. Widespread direct auditing of the code, as open source proponents expect, is more costly and less effective. In addition, many race conditions, which allow "time of check/time of use" attacks, could also be eliminated with modern programming languages support for exception handlers. In short, inspections facilitated by open-source development are not directing a useful grease to where it could have a significant impact.

Anticipating the Next Problem. The trend towards deploying highly extensible system architectures provides avenues for attack enabled more by the absence of security functionality than by the presence of bugs. Web browsers allow "plug-ins" so that additional types of objects can be downloaded; operating systems support "plug-and-play" for adding new hardware; office applications allow user-defined macros to accompany documents or spread-sheets, creating new abstractions and operations. And software producers have much to gain from deploying extensible architectures. First, customers are retained because new hardware and information technologies now can be embraced incrementally. Second, extensible architectures allow software developers to target their products to specific software layers and thus leverage development costs across multiple platforms and diverse user needs.

We can protect a base system from hostile extensions by employing the Principle of Least Privilege and supporting a notion of "least privilege" that depends on current and past actions, context, and so on. Unfortunately, this sort of fine-grained access control is not supported in today's systems. We also today have no way to obtain for an extension a formulation of its "least privilege" security policy—expecting users, who today have difficulty managing a small number of file access-control bits, to control the significantly larger collection of bits specifying "least privilege" is simply not realistic. The problem is missing security functionality, some of which we know how to implement and some of which we do not. And this problem, whose solution is critical for solving the malicious code problem, is independent of whether open-source or closed-source development practices are used.

Assurance Idealism versus Realism. Not only must a system do what is expected (and no more or less), but we must somehow have confidence that this will be so. One way to obtain assurance in a software system is by analyzing the code; an alternative is analyzing the process used in creating the code. Most would agree that assurance obtained by direct analysis of code is more convincing than assurance obtained by analyzing facets of the construction process. But most would also agree that cost and effectiveness play the big role in defining an analysis method's practicality.

By definition, open-source software development allows direct analysis of the code by everyone but rules out control of the construction process. This is a problem if existing methods for direct analysis are not practical but controlling construction is. For example, one hopes to reduce the chances of shipping systems that contain Trojan horses through careful hiring practices and attention to employee morale, an approach that is unavailable with open-source development. (Suggesting that software be inspected for Trojan horses misses the point.) So, at least for security, the inability to control the construction process could be a drawback of open-source software.

Closed-source software development allows both direct analysis and control of construction, but only by the software producer or some partner. Thus, customers must trust the software developer for claims about assurance. Assurance is not valued in today's market, so developers do not have much incentive to devote resources here—our reluctance to trust software developers is justifiable. But the market could change, and if it does, a closed-source software producer would have the financial incentive in addition to the ability to perform both direct and indirect analysis. One should be willing to trust a developer whose profits and very existence depended on being trustworthy.

Concluding Remarks. Open-source software development has much to recommend it. Having system source code be public might well lead to systems that function more reliably and support functionality that users seek. Today's security woes, however, are not dominated by the existence of bugs that might be discovered by open-source developers studying system source code. Use of decent programming languages would have a far bigger impact. And today's missing security functionality is not something that we even know how to support. So, open-source software development contributes little to the real security problems of today's systems. Moreover, should security become a priority, the market would respond. Then, closed-source development practices could employ a broader range of assurance methods and would have a plausible business model, where open-source does not.

Acknowledgments Discussions with Peter G. Neumann started me thinking about these issues. I am grateful to Yaron Minsky, Greg Morrisett, Andrew Myers, and Bob Constable for their comments and criticisms as my thoughts and writing on this subject has matured.