Robust Nonproprietary Software

Peter G. Neumann
Principal Scientist, Computer Science Lab
SRI International, Menlo Park CA 94025-3493
Neumann@csl.sri.com, http://www.csl.sri.com/neumann, 1-650-859-2375

Our ultimate goal here is to be able to develop robust systems and applications that are capable of satisfying serious requirements, not merely for security but also for reliability, fault tolerance, human safety, and survivability in the face of a wide range of realistic adversities – including hardware malfunctions, software glitches, inadvertent human actions, massive coordinated attacks, and acts of God. Also relevant are additional operational requirements such as interoperability, evolvability and maintainability, as well as discipline in the software development process.

Despite all our past research, development of commercial systems is decidedly suboptimal with respect to meeting stringent requirements. This brief paper examines the applicability of some alternative paradigms.

To be precise about our terminology, we distinguish here between *black-box* (that is, closed-box) systems in which source code is not available, and *open-box* systems in which source code is available (although possibly only under certain specified conditions). Black-box software is often considered as advantageous by vendors and believers in security by obscurity. However, black-box software makes it much more difficult for anyone other than the original developers to discover vulnerabilities and provide fixes therefor. It also hinders open analysis of the development process itself (which is something many developers are happy to hide). Overall, it can be a serious obstacle to having any unbiased confidence in the ability of a system to fulfill its requirements (security, reliability, safety, etc., as applicable).

We also distinguish here between *proprietary* and *nonproprietary* software. Note that open-box software can come in various proprietary and nonproprietary flavors.

Examples of nonproprietary open-box software are increasingly found in the Free Software Movement (such as the Free Software Foundation's GNU system with Linux) and the Open Source Movement, although discussions of the distinctions between those two movements and their respective nonrestrictive licensing policies are beyond the scope of this brief analysis. In essence, both movements believe in and actively promote unconstrained rights to modification and redistribution of open-box software [2].

The benefits of nonproprietary open-box software include the ability of outside good guys to carry out peer reviews, add new functionality, identify flaws, and fix them rapidly – for example, through collaborative efforts involving people widely dispersed around the world. Of course, the risks include increased opportunities for evil-doers to discover flaws that can be exploited, or to insert trap doors and Trojan horses into the code.

A question for this panel is what are the roles of open-box software in developing robust systems, in light of (for example) the Internet, typically flawed operating systems, vulnerable system embeddings of strong cryptography, and the presence of mobile code. An architectural subquestion involves where trustworthiness must be placed to minimize the amount of critical code and to achieve robustness in the presence of the specified adversities.

Will open-box software really improve system security? My answer is *not by itself, although the potential is considerable.* Many other factors must be considered. Indeed, many of the problems of black-box software can also be present in open-box software, and *vice versa* (for example, flawed designs, the risks of mobile code, a shortage of gifted system administrators, and so on). In the absence of significant discipline and inherently better system architectures, opportunities may be even more widespread for insertion of malicious code in the development process, and for uncontrolled subversions of the operational process.

We face the basic conflict between (a) security by obscurity to slow down the adversaries, and (b) openness to allow for more thorough analysis [3] and collaborative improvement of critical systems – as well as providing a forcing function to inspire improvements in the face of discovered attack scenarios. Ideally, if a system is meaningfully secure, open specifications and open-box source should not be a significant benefit to attackers, and the defenders might be able to maintain a competitive advantage! For example, this is the principle behind using strong openly published cryptographic algorithms – for which analysis of algorithms

and their implementations is very valuable, and where only the private keys need to be hidden. Other examples of obscurity include tamperproofing and obfuscation. Unfortunately, many existing systems tend to be poorly designed and poorly implemented, with respect to incomplete and inadequately specified requirements. Developers are then at a decided disadvantage, even with black-box systems. Besides, research initiated in a 1956 paper by Ed Moore [1] reminds us that purely external *(Gedanken)* experiments on black-box systems can often determine internal state details.

Behavioral system requirements such as safety, reliability, and real-time performance cannot be realistically achieved unless the systems are adequately secure. It is very difficult to build robust applications based on proprietary black-box software that is not sufficiently trustworthy.

Further 1956 papers, by Moore, Claude Shannon, and John von Neumann, showed how to construct reliable components out of less reliable components. Later work on correct behavior despite some number of arbitrarily perverse Byzantine faults followed along those lines. In that context, building a fault-tolerant silk purse out of less robust sow's ears is indeed possible in some cases. But constructing more trustworthy secure systems out of less trustworthy subsystems does not seem realistic when the underlying components are compromisible, despite efforts such as wrapper technology and firewall isolation.

Whenever achieving security by obscurity is not the primary goal, there seem to be strong arguments for open-box software that encourages open review of requirements, designs, specifications, and code. Even when obscurity is deemed necessary, some wider-community open-box approach is desirable. For software and for system applications in which security can be assured by other means and is not compromisible within the application itself, the open-box approach has particularly great appeal. In any event, it is always unwise to rely *solely* on obscurity.

So, what else is needed to achieve trustworthy robust systems that are predictably dependable? The first-level answer is the same for open-box systems as well as closed-box systems: serious discipline throughout the development cycle and operational practice, use of good software engineering, rigorous repeated evaluations of systems in their entirety, and enlightened management, for starters.

A second-level answer involves inherently robust and secure evolvable interoperable architectures that avoid excessive dependence on untrustworthy components. One such architecture involves thin-client user platforms with minimal operating systems, where trustworthiness is bestowed where it is essential – typically, in servers, firewalls, code distribution paths, nonspoofable provenance for critical software, cryptographic coprocessors, tamperproof embeddings, preventing denial-of-service attacks, runtime detection of malicious code and deviant misuse, *etc.* [4].

A third-level answer is that there is still much research yet to be done (such as on realistic compositionality, inherently robust architectures, and open-box business models), as well as more efforts to bring that research into practice. Effective technology transfer seems much more likely to happen in open-box systems.

Nonproprietary open-box systems are not a panacea. However, they have potential benefits throughout the process of developing and operating critical systems. Impressive beginnings already exist. Nevertheless, much effort remains in providing the necessary development discipline, adequate controls over the integrity of the emerging software, system architectures that can satisfy critical requirements, and well documented demonstrations of the benefits of open-box systems in the real world. If nothing else, open-box successes may have an inspirational effect on commercial developers, who can rapidly adopt the best of the results. But I like the possibilities for coherent community cooperation, and have considerable hope for nonproprietary open-box software.

References

[1] E.F. Moore, Gedanken Experiments on Sequential Machines, *Automata Studies*, Annals of Mathematical Studies, 34, C.E. Shannon and J. McCarthy, eds., Princeton University Press, 1956. pp. 129-153.

[2] The Free Software Foundation Website is `http://www.gnu.org`, and contains software, projects, licensing procedures, *etc.:* The Open Source Movement Website is `http://www.opensource.org/`, which includes Eric Raymond's "The Cathedral and the Bazaar" and the Open Source Definition.

[3] Analytic tools for open-box source code include Crispin Cowan's StackGuard (`http://immunix.org`), David Wagner's buffer overflow analyzer (`http://www.cs.berkeley.edu/~daw/papers/`), @Stake's L0pht security review analyzer slint (`http://www.l0pht.com/slint.html`) and RST's ITS4 function-call analyzer for C and C++ code (`http://www.rstcorp.com/its4/`).

[4] The U.S. Army Research Laboratory (ARL) has supported my work on survivable systems, under contract DAKF11-97-C-0020. See `http://www.csl.sri.com/neumann` for a report for ARL together with a course taught in the fall of 1999 on developing robust systems. That Website also contains a paper on single-level multilevel-secure systems, N.E. Proctor and P.G. Neumann, Architectural Implications of Covert Channels, *Proceedings of the Fifteenth National Computer Security Conference*, Baltimore, Maryland, October 13–16, 1992, pp. 28–43, very much in the spirit of the thin-client architecture noted above.