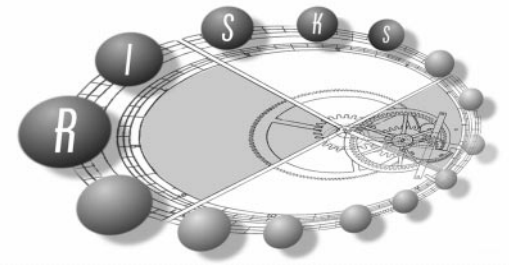

Peter G. Neumann

Robust Open-Source Software

Closed-source proprietary software, which is seemingly the lifeblood of computer system entrepreneurs, tends to have associated risks:

- Unavailability of source code reduces on-site adaptability and repairability.
- Inscrutability of code prohibits open peer analysis (which otherwise might improve reliability and security), and masks the reality that state-of-the-art development methods do not produce adequately robust systems.
- Lack of interoperability and composability often induces inflexible monolithic solutions.
- Where software bloat exists, it often hinders subsetting.
- Proprietary interface standards complicate system integration.

A well-known (but certainly not the only) illustration of these risk factors is Windows NT 5.0. It reportedly will have 48 million lines of source code in the kernel alone, plus 7.5 million lines of associated test code. Unfortunately, the code on which security, reliability, and survivability of system applications depend is essentially all 48M lines plus application code. (Recall the divide-by-zero in an NT application that brought the Yorktown Aegis missile cruiser to a halt [Risks Forum *19*, 88 (Jul. 22, 1998)].) In critical applications, an enormous amount of untrustworthy code may have to be taken on faith.

Open-source software offers an opportunity to surmount these risks of proprietary software. "Open Source" is registered as a certification mark, subject to the conditions of The Open Source Definition (www.opensource.org/osd.html), which has various explicit requirements: unrestricted redistribution; distributability of source code; permission for derived works; constraints on integrity; nondiscriminatory practices regarding individuals, groups, and fields of endeavor; transitive licensing of rights; context-free licensing; and noncontamination of associated software. For background, see the opensource.org Web site, which cites conformant examples. Additional useful sources include the Free Software Foundation (www.gnu.org). The Netscape browser (an example of open, but proprietary software), Perl, Bind, the Gnu system with Linux, Gnu Emacs, GCC, to name a few, are further examples of what can be done. Also, Diffie-Hellman is now in the public domain.

In many critical applications, we desperately need operating systems and applications that are meaningfully *robust*, where "robust" is an intentionally inclusive term embracing meaningful security, reliability, availability, and system survivability, in the face of a wide and realistic range of potential adversities—which might in some cases include hardware faults, software flaws, malicious and accidental exploitation of systemic vulnerabilities, environmental hazards, unfortunate animal behaviors, and so forth.

We need significant improvements on today's software, both open-source and proprietary, in order to overcome myriad risks (see the RISKS archives (catless.ncl.ac.uk/Risks/) or my Illustrative Risks document (www.csl.sri.com/~neumann/). When commercial systems are not adequately robust, we should consider how sound open-source components might be composed into demonstrably robust systems. This requires an international collaborative process, open-ended, long-term, far-sighted, somewhat altruistic, incremental, and with diverse participants from different disciplines and past experiences. Pervasive adherence to good development practice is also necessary (suggesting better teaching). The process also needs some discipline, in order to avoid rampant proliferation of incompatible variants. Fortunately, there are already some very substantive efforts to develop, maintain, and support open-source software systems, with significant momentum. If those efforts can succeed in producing demonstrably robust systems, they will also provide an incentive for better commercial systems.

We need techniques that augment the robustness of less robust components, public-key authentication, cryptographic integrity seals, good cryptography, trustworthy distribution paths, trustworthy descriptions of the provenance of individual components and who has modified them. We need detailed evaluations of components and the effects of their composition (with interesting opportunities for formal methods). Many problems must be overcome, including defenses against Trojan horses hidden in systems, compilers, and evaluation tools—especially when perpetrated by insiders. We need providers who give real support; warranties on systems today are mostly very weak. We need serious incentives including funding for robust open-source efforts. Despite all the challenges, the potential benefits of robust open-source software are worthy of considerable collaborative effort. C

Peter G. Neumann (www.csl.sri.com/neumann/) chairs the ACM Committee on Computers and Public Policy.

PAUL WATSON