Alternatives to Proprietary Closed-Source Software

Dr. Peter G. Neumann
Principal Scientist
Computer Science Lab
SRI International, Menlo Park
California USA 94025-3493
Telephone 1-650-859-2375
E-mail Neumann@CSL.sri.com
http://www.csl.sri.com/neumann

I-4 Forum 42
27 February 2001

Abstract

We summarize the advantages and disadvantages of proprietary closed-source software, and consider various alternatives, particularly with respect to the need for highly secure systems.

Background

See my position paper "Robust Nonproprietary Software" for the 2000 IEEE Symposium on Security and Privacy [1], and my U.S. Army Research Lab final report on survivable systems and networks [2]. References and relevant on-line sites are given in the final slides.

Fundamental System Needs

- Critical information systems and their applications must predictably satisfy stringent requirements such as security, reliability and fault tolerance, safety, robustness, and survivability in the face of many adversities, interoperability, evolvability, maintainability, ...
- Stringent requirements cannot be achieved without many factors, such as good development practice, good operational practice, etc.
- Security is a holistic problem.

Some Generic Problems in Software Development and Use

• Developing robust systems is inherently difficult.

• Today's "Best Practices" are inadequate and poorly applied.

• Software engineering is poor, e.g., bounds/buffer overflows.

• The *Common Criteria* approach is incomplete and unwieldy, but better than its predecessors. Protection profiles are likely to be incomplete.

• Operational management is difficult and riskful.

• These problems are ubiquitous.

System Realities Today

• Distributed and networked systems are riskful. Embedded systems are increasingly dependent on other systems, rather than standalone.

• Personal-computer operating systems are typically flawed, bloated, inflexible, difficult to administer, not suitable for critical applications. Server OSs and firewalls are often misconfigured. Network protocols are inadequate. There are many risks.

• "Defense in Depth" is more like "Weakness in Depth". There are too many weak links.

System Failures and Misuses

• Internet Worm: flawed replication strategy, closed source! (cf. 1980 ARPAnet & 1990 AT&T collapses)

• System/network exploitations: denial of service attacks, Website hacks (DoJ, CIA, USAF, NASA, Cloverdale Pentagon, Army, FBI, Senate, etc.), Citibank: flawed designs, implementations, operations

• Poor Device security: DVD CSS, GSM, CMEA crypto, smartcards; security by obscurity

• John Deutch, Wen Ho Lee, and Dept of State laptops: security by classification and faith

Proprietary-System Problems

• Many proprietary systems are not capable of satisfying critical requirements.

• *Black-box* systems (i.e., closed-box, in which source code is unavailable) hinder open analysis of system development processes and the resulting software quality, impede system integration, and prevent urgent on-site self-remediation.

• Lack of interoperability and composability often encourages inflexible monolithic solutions.

More Black-Box Problems: Need for Reverse-Engineering

- Interoperability, local code patching for flaw removal, maintenance, and other constructive purposes conflict with the original World Intellectual Property Organization (WIPO) Copyright Treaty prohibitions against reverse engineering, although some of those restrictions have been somewhat relaxed.
- The U.S. 1998 Digital Millennium Copyright Act is overly restrictive. So is the pending U.S. Uniform Computer Information Transactions Act (UCITA).

UCITA Allows Vendors To

- Have total immunity from liability
- Bar use of proprietary interfaces
- Inhibit constructive reverse engineering, even for debugging and patching
- Install trapdoors that enable them to disable installed software remotely!
- Forbid publication of criticism
- Discourage open-source

Already law in Virginia and Maryland, on dockets in Arizona, Oklahoma, Delaware and Texas.

Black-Box Software: Benefits for System Developers

- Black-box code masks intellectual property, and impedes development of competitors' systems and dependent applications, hinders interoprability.
- Proprietary black-box business models are well understood.
- Consumer retention is enhanced inertially, because switching is more difficult.
- For secure systems, attackers may be slowed down somewhat by security through obscurity – although this is debatable.

Black-Box Software: Potential Benefits for Users

- For proprietary systems, the identified proprietor is the supposed target for remediation, lawsuits, etc. (in the absence of UCITA). (However, this is not necessarily a benefit, as remediation is often slow and lawsuits are even slower!)
- For black-box systems, there are very few significant benefits for users that cannot also be achieved with available source code, except possibly the delaying effects of security by obscurity.

Terminology

• *Open-box* code, i.e., *source-available* and *source-alterable,* is the antithesis of *black-box* code. Many examples of open-box software are found in the Open-Source Movement and the Free Software Movement (see next slides and cited Websites), with various distribution licenses. (Note: open-box software may or may not be proprietary.)

• Open Source and Free Software are not equivalent, although analysis of the differences is beyond the scope of this discussion.

The Free Software Movement's Free Software Foundation (FSF); `http://www.gnu.org`

In FSF, *free* implies *freedom to copy* and *freedom to change,* not necessarily *free of charge.* FSF incentivizes collaborative efforts and continual improvements. Founded 1984.

• The FSF General Public License (GPL) enforces *copyright* plus *copyleft,* where copyleft requires that redistribution (with or without change) must not restrict freedom to further copy and change.

Open Source Movement's Open Source Definition (OSD) `http://www.opensource.org/osd.html`

• Unrestricted redistribution
• Distributability of source code
• Permission for derived works
• Constraints on integrity
• Nondiscriminatory practices
• Transitive licensing of rights
• Context-free licensing
• No adverse affects on associated software

Open-Box Examples, e.g., Free and Open-Source Software

• GPL-ed: The GNU System with Linux (GNU Emacs, GCC, Gnome 2.0, Ghostview, GNUscape Navigator, gzip, Java packages, etc.), Free VSD; not quite GPL-ed software (Perl); non-GPL free software (Free BSD, X windows, Apache, $\LaTeX$, Mozilla, Netscape JavaScript ...); Open BSD, Net BSD, Hyperlatex, Eazel's Linux graphical shell, ...

• Other licenses: MPL, QPL, ...

Open-Box Potentials 1

• Extensive peer review is easy and normal, and amenable to academics and other researchers. Many eyeballs (Eric Raymond)
• Open design is valuable. So are open requirements.
• Peer analysis is capable of finding flaws and generating fixes rapidly.
• Open-box software is potentially more readily capable of incremental evolution.
• Such software is often developed altruistically and is less motivated by short-term cost-cutting.

Open-Box Potentials 2

• Users and administrators are potentially in greater control, because they may be able to obtain fixes and new features readily.
• People other than the original developers can add significant value, e.g., making systems more robust.
• Open collaboration is easier.
• Software quality can be very high.
• Already 7000 active projects, 750,000 participants, 150,000 hardcore developers (says Raymond)

Analytic Benefits of Open-Box Code

The availability of source code for analysis (even if it is proprietary) enables application of analytic tools such as
• Crispin Cowan, StackGuard `http://immunix.org`
• David Wagner et al., Berkeley buffer overflow analyzer approach `http://www.cs.berkeley.edu/~daw/papers/`
• L0pht (now part of @Stake), slint `http://www.l0pht.com/slint.html`
• Reliable Software, ITS4 for C, C++ `http://www.rstcorp.com/its4/`

Open-Box Problems

• Many of the problems of black-box software are also applicable to open-box software, for example, the risks of mobile code.
• Detected flaws may not be reported.
• Opportunities may be more widespread for insertion of malicious code (e.g., Trojan horses) during development, and for operational subversions.
• Management may be needed across organizations, and is itself difficult and possibly riskful.

Open vs Closed Analysis: Seemingly Contradictory Views

• Easier access by adversaries to available source implies less operational security, because it is easier to find exploitable flaws in vulnerable systems.

• Open box should be particularly important for the analysis and improvement of life-critical and ultra-reliable systems. Also, if a system is meaningfully secure, open specs and available source should not be of less benefit to attackers, which could give defenders a competitive advantage (for a change).

Available Source Is Only Part of What Is Needed; There's More.

• Open-box source code shares many generic problems with black-box source.

• Much more is needed to make open-design open-box systems robust, trustworthy, and predictably dependable – notably discipline throughout the life cycle.

• See my Website for ARL report on developing survivable systems and networks:
`http://www.csl.sri.com/neumann`

Generic Desiderata

• Discipline in development, software engineering, distribution, operation, evolution, evaluations, education, training, ...

• Inherently robust secure evolvable interoperable architectures

• Responsible operational support and configuration control

• Open standards for software, interfaces, composability, interoperability, distribution

• Contracts, liabilities, incentives: compliance bonuses, noncompliance penalties

• Sound business models for nonproprietary open-box software

Robust Architectures

• Architectures that avoid excessive dependence on untrustworthy components

• Thin-client user platforms with minimal operating systems, where trustworthiness is required only where essential

• Trustworthy servers, firewalls, distribution paths for software, provenance on all critical software; nontrivial user authentication, bilateral peer authentication, aggressive resistance to denials of service, better protocols, ...

More on Robust Architectures

- Nonsubvertible implementations of cryptography, used pervasively, including cryptographic integrity
- Run-time detection of malicious code and misuse
- Wireless applications and mobile code add some stringent further requirements.

See the DARPA CHATS BAA (Composable High Assurance Trusted Systems strongly encouraging open-source software): http://www.darpa.mil/baa/BAA01-24.htm

Conclusions

- Nonproprietary open-box software (e.g., Free Software and Open-Source) is not a panacea, but has huge potential, with discipline and well-documented successes. (Discipline is similarly needed for black-box software, but is often lacking.)
- Open-box source could be particularly promising in efforts to develop dependable critical systems.
- Open-box successes can be an incentive to black-box developers, some of whom are already exploring such alternatives.

References

1. Panel position papers by Steve Lipner, Gary McGraw, Peter Neumann, Fred Schneider, *Proc. 2000 IEEE Symposium on Security and Privacy,* Berkeley CA, 15-17 May 2000. My paper is at `http:/www.csl.sri.com/neumann/ieee+.ps` (and `/ieee+.pdf`).

2. Peter G. Neumann, Practical Architectures for Survivable Systems and Networks, SRI final report for Army Research Lab, 30 June 2000. `http:/www.csl.sri.com/survivable.html` for browsing, `survivable.ps` and `survivable.pdf` for printing.

A Few On-Line References

- Peter G. Neumann: papers, reports, testimonies, survivability course notes, RISKS, Illustrative Risks, CACM Inside Risks, etc.: `http://www.csl.sri.com/neumann`
- Free Software Foundation: software, philosophy, projects, licenses, etc.: `http://www.gnu.org`
- Eric Raymond: Cathedral & Bazaar; Hallowe'en Documents `http://www.opensource.org/`: "Open Source promotes software reliability and quality by supporting independent peer review and rapid evolution of source code."

BIOGRAPHICAL BACKGROUND

Peter G. Neumann is a Principal Scientist in the Computer Science Laboratory at SRI (where he has been since 1971), concerned with computer system survivability, security, reliability, human safety, and high assurance. He is the author of *Computer-Related Risks,* Moderator of the ACM Risks Forum (comp.risks), Chairman of the ACM Committee on Computers and Public Policy, and Associate Editor of the CACM for the Inside Risks column. He founded and 19 years edited the ACM SIGSOFT *Software Engineering Notes.* He is now a member of the U.S. General Accounting Office Executive Council on Information Management and Technology. See http://www.CSL.sri.com/neumann for U.S. Senate/House and California Assembly testimonies, reports, RISKS, papers, slides, etc.

Neumann taught at the Technische Hochschule Darmstadt in 1960, Stanford University in 1964, the University of California at Berkeley in 1970-71, and

most recently a course on survivable systems and networks at the University of Maryland in the fall of 1999 (see my Website for notes).

Neumann is a Fellow of the American Association for the Advancement of Science, the ACM, and the Institute of Electrical and Electronics Engineers (of which he is also a member of the Computer Society). He has received the ACM Outstanding Contribution Award for 1992, the first SRI Exceptional Performance Award for Leadership in Community Service in 1992, the Electronic Frontier Foundation Pioneer Award in 1996, the ACM SIGSOFT Distinguished Service Award in 1997, and the CPSR Norbert Wiener Award for in October 1997, for "deep commitment to the socially responsible use of computing technology."

Peter G. Neumann, Computer Science Laboratory, SRI International 333 Ravenswood Ave., Menlo Park CA 94025-3493 Telephone 650-859-2375, FAX 650-859-2844, neumann@csl.sri.com. http://www.csl.sri.com/neumann.html