Today’s invasion of software into every aspect of daily life cannot be questioned. The problem, however, is that software, which is supposed to improve the quality of human life, can also damage it. Software is supposed to do tasks that humans cannot do, prefer not to do, or cannot do as quickly or efficiently. As we increasingly relinquish control over our everyday lives to software, the risk of becoming dependent on systems that do not perform correctly increases. Worse, knowledge that we are living under the shroud of these risks is often absent, making us more vulnerable than we know.

Therefore, industry currently faces a “software versus people” showdown. In the past three years, terms such as “cyber warfare,” “information terrorism,” “information warfare,” and “information survivability” have become part of our vocabulary—even entering the mainstream media. The reason is simple: information controls many of the critical services that people, corporations, and governments depend on.

The losses that could result from software that behaves in undesirable ways stem from a variety of human-caused problems. Some problems are simply negligent development practices that lead to defective software. Les Hatton exposed the fact that defect densities have remained fairly constant during the past 20 years for all types of software: 6 to 30 faults per thousand source lines of code (KSOLOC). And Business Week
The Internet and public phone system (upon which the Internet sits) provide an information highway that was not designed to thwart “bad guys.”

Malicious IT

- The “Malicious IT Roundtable” contains a freewheeling discussion of information security and the associated risks, with a focus on security policy. The participants are drawn from many countries, representing industry, government labs, and academe.
- Gary McGraw’s and Greg Morrisett’s article, “Attacking Malicious Code: A Report from the Infosec Research Council,” reports on the results of meetings of the Malicious Code Study Group. After defining malicious code, the article discusses the problems introduced by malicious code and defenses against malicious code.
- Thomas Bowen’s article, “Remediation of Application-Specific Security Vulnerabilities at Runtime,” has a technical focus and describes a solution to the application vulnerability problem, based on monitoring and changing an application’s behavior by intercepting the system calls it requests.
- “Security Domains: Key Management in Large-Scale Systems” by John R. Michener and Tolga Acar discusses implementation and management of security domains. They provide a general architecture and initial design to enable the use of shared keys.
- “Defending Yourself—The Role of Intrusion Detection Systems,” by John McHugh, Alan Christie, and Julia Allen, discusses the role of intrusion detection systems as a defensive measure. There is also an overview of commonly used intrusion detection techniques and representative examples of intrusion detection products and tools.
- “Statically Scanning Java Code for Security Vulnerabilities,” by John Viega, Tom Mudge, Gary McGraw, and Edward W. Felten, describes a prototype tool that statically scans Java source code for potentially insecure code. The article discusses the problems introduced by malicious code and defenses against malicious code.

Online (6 December 1999) wrote: “According to the US DoD and the SEI, there are typically 5–15 flaws in every 1,000 lines of code.” Regardless of which numerical range is closer to the industry’s true “average defect rate,” large commercial systems clearly have large numbers of defects.

Security vulnerabilities that result from negligent development practices (for example, commercial Web browsers allowing unauthorized individuals to access confidential data) are likely to be discovered by rogue individuals with malicious intentions. Other security vulnerabilities are deliberately and maliciously programmed into software (such as logic bombs, Trojan Horses, attack scripts, and Easter eggs), and these vulnerabilities are often referred to as malicious code. Malicious code is simply any software functionality that has been added, deleted, or modified to intentionally cause harm. Those types of vulnerabilities are the focus of this special topic in IEEE Software—ones that represent situations in which people can knowingly develop and execute software solely for the purpose of harming others.

Many of the problems that we face today are a result of our continued reuse of systems (like Unix) that were never designed to be secure. Unix was designed to allow computers to talk, with the implicit assumption that computers that communicated were trustworthy.

Furthermore, we must recognize that the Internet and public phone system (upon which the Internet sits) provide an information highway that also was not designed to thwart “bad guys.” As a result, today we rely on an infrastructure that enables rogue individuals and nations to remotely attack information assets. With 67,000 additional people rumored to gain access to the Internet daily, the list of potential victims and attackers increases.

Thus the vulnerabilities passed on to society from defective and malicious software are real. In fact, news about this has made it to the highest levels of government within the US, as evidenced by a recent New York Times report:

Washington, July 28 (Bloomberg)—The administration of US President Bill Clinton wants the FBI to oversee an extensive computer monitoring system to protect the nation’s crucial data networks from intruders, the New York Times reported, citing a draft of the plan. It calls for a sophisticated software system to monitor activities on nonmilitary government networks and a separate system to track networks used in the banking, telecommunications, and transportation industries. Critics of the plan charge that it could lead to a surveillance infrastructure with great potential for misuse, the newspaper said.

And from the Office of the White House Press Secretary, note the following statements from the 22 May 1998 Presidential Decision Directive 63, titled “The Clinton Administration’s Policy on Critical Infrastructure Protection”:

Every department and agency of the Federal Government shall be responsible for protecting its own critical infra-
structure, especially its cyber-based systems. Every department and agency Chief Information Officer (CIO) shall be responsible for information assurance. Every department and agency shall appoint a Chief Infrastructure Assurance Officer (CIAO) who shall be responsible for the protection of all of the other aspects of that department’s critical infrastructure. The CIO may be double-hatted as the CIAO at the discretion of the individual department. These officials shall establish procedures for obtaining expedient and valid authorizations to allow vulnerability assessments to be performed on government computer and physical systems. The Department of Justice shall establish legal guidelines for providing for such authorizations.

In this special topic in IEEE Software, the authors and roundtable participants reflect the concerns in malicious information technology we’ve outlined here and suggest steps towards solving the growing risks associated with malicious information technology (see the “Malicious IT” box for a thumbnail description of the articles). Our objective is to heighten the awareness of Software’s readers regarding this growing problem and to identify resources in formulating action plans and solutions.

### About the Editors

**Nancy Mead** is the team leader for the Survivable Network Analysis team as well as a senior member of the technical staff in the Networked Systems Survivability Program at the Software Engineering Institute. The CERT Coordination Center is a part of this program. She is also a faculty member in the Master of Software Engineering and Master of Information Systems Management programs at Carnegie Mellon University. She is involved in the study of survivable systems architectures and the development of professional infrastructure for software engineers. She received her PhD in mathematics from the Polytechnic Institute of New York, and a BA and an MS in mathematics from New York University. She is also a senior member of the IEEE and the IEEE Computer Society and is also a member of the ACM. Contact her at the Software Engineering Inst., 5000 Forbes Ave., Pittsburgh, PA 15213; nrm@sei.cmu.edu.

**Jeffrey Voas** is cofounder and chief scientist of Capital (formerly Reliable Software Technologies). He has a BSE in computer engineering from Tulane and MS and PhD degrees in computer science from the College of William and Mary. He is the coauthor (with Gary McGraw) of *Software Fault Injection: Inoculating Programs against Errors* (John Wiley & Sons, New York, 1997). He serves on the editorial board for IEEE Software and IEEE IT Pro. Contact him at 21351 Ridgeway Cirle, Suite 400, Dulles, VA 20166; jmvoas@rstcorp.com.