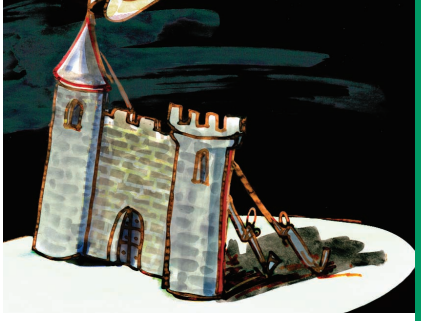

# Building Software Securely from the Ground Up

**Anup K. Ghosh,** *Cigital*
**Chuck Howell,** *MITRE*
**James A. Whittaker,** *Florida Institute of Technology*

As software professionals first and computer security researchers second, we believe that most computer security articles published in industry magazines focus on fixing security problems after the fact. Indeed, many of these "fixes" tend to be point solutions that address symptoms of a much larger problem and fail to address the root causes of the real computer security problem. Our assessment of the state of the industry motivated us to pull together a special theme issue for *IEEE Software* focusing on how to build software systems securely from the ground up.

Of course, we're not the only people who believe that systems must be built this way. Richard Clarke, the US President's Special Advisor for Cyberspace Security, admonished our industry at the Business Software Alliance's Global Technology Summit in Washington, D.C., on 4 December 2001:

> *To start, members of the IT industry must build information security into their products at the point of development and not treat it as an afterthought.*

## Break the cycle

The types of approaches and point solutions advocated by computer security professionals to date have aimed at system administrators, chief information officers, and other personnel involved in system infrastructure management. These solutions usually focus on addressing an enterprise's security defenses (such as firewalls, routers, server configuration, passwords, and encryption) rather than on one of the key underlying causes of security problems—bad software.

Although point solutions are often necessary, they are inadequate for securing Internet-based systems. Point solutions are temporary barriers erected to stem the tide of attacks from unsophisticated attackers. In a sense, point solutions are simply bandages on a broken machine; they might stop some bleeding, but the wound underneath still festers. Fundamentally, we believe the problem must be fixed at its core by building secure, robust, survivable software systems. For this issue, we solicited articles from software researchers and professionals to provide guidance and innovation for building software systems to be resistant to attack.

We start from the premise that most se-


0740-7459/02/$17.00 © 2002 IEEE

curity breaches in practice are made possible by software flaws. Statistics from the Software Engineering Institute's CERT Coordination Center on computer security incident reports support this assertion (see the "Further Reading" sidebar). But engineering secure and robust software systems can break the penetrate-and-patch cycle of software releases we have today (see the "Penetrate and Patch Is Bad" sidebar).

The key goal of this issue is to encourage a deeper understanding of how security concerns should influence all aspects of software design, implementation, and testing. A notorious example of poor software implementation is the buffer overflow vulnerability. Known for decades, and very troublesome in networked systems, it continues to be introduced into new software at an alarming rate, due in part to software development habits that trace back to isolated systems where such flaws had few security implications.

Software designers in a networked world cannot pretend to work in isolation. People are a critical part of the full software secu-

## Penetrate and Patch Is Bad

**Gary McGraw**

Many well-known software vendors don't understand that security is not an add-on feature. They continue to design and create products at alarming rates, with little attention paid to security. They start to worry about security only after someone publicly (and often spectacularly) breaks their products. They then rush out a patch instead of realizing that adding in security from the start might be a better idea. This sort of approach won't do in e-commerce or other business-critical applications.

We should strive to minimize the unfortunately pervasive penetrate-and-patch approach to security and avoid desperately trying to come up with a fix to a problem that attackers are actively exploiting. In simple economic terms, finding and removing bugs in a software system before its release is orders-of-magnitude cheaper and more effective than trying to fix systems after release.[1]

The penetrate-and-patch approach to security presents many problems:

- Developers can only patch problems they know about. Attackers usually don't report the problems they find to developers.
- Vendors rush out patches as a result of market pressures and often introduce new problems into the system.
- Patches often only fix a problem's symptoms; they do nothing to address the underlying cause.
- Patches often go unapplied, because system administrators tend to be overworked and often do not wish to make changes to a system that "works." In many cases, system administrators are generally not security professionals.

Designing a system for security, carefully implementing the system, and testing the system extensively before release presents a much better alternative.

The fact that the existing penetrate-and-patch approach is so poorly implemented is yet another reason why we must change it. In the December 2000 *Computer* article "Windows of Vulnerability: A Case Study Analysis," Bill Arbaugh, Bill Fithen, and John McHugh discuss a life-cycle model for system vulnerabilities that

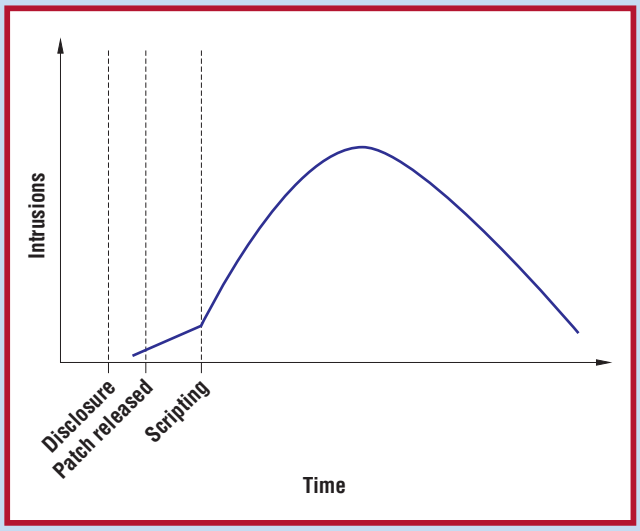


**Figure A. An average curve of the number of intrusions by a security bug over time.**

emphasizes how big the problem is.[2] Data from their study shows that intrusions increase once a vulnerability is discovered, the rate continues to increase until the vendor releases a patch, and exploits continue to occur even after the patch is issued (sometimes years after). Figure A is based on their data.

It takes a long time before most people upgrade to patched versions because most people upgrade for newer functionality or the hope of more robust software or better performance, not because they know of a real vulnerability.

### References

1. F. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*, 2nd ed., Addison-Wesley, Reading, Mass., 1995.
2. B. Arbaugh, B. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," *Computer*, vol. 33, no. 12, Dec. 2000, pp. 52–59.

**Gary McGraw** is Cigital's chief technology officer and a noted authority on mobile-code security. His research interests include software security and software risk management. He received a BA in philosophy from the University of Virginia and a PhD in cognitive science and computer science from Indiana University. Contact him at gem@cigital.com.

rity equation, and software that makes unrealistic or unreasonable security-related demands on users (for example, requiring them to memorize too many passwords that change too often) is software whose security will inevitably be breached.

Finally, although individuals exploiting flawed software cause most security breaches today, a more ominous emerging threat is malicious code that is deliberately written to exploit software flaws. Examples are the Code Red and Nimda worms and their variants, which can spread on an Internet scale in Internet time. To effectively address current and future problems in computer security, we must build software securely from the ground up.

## The articles

We are fortunate to have received numerous outstanding article contributions for this issue—clearly speaking to the importance of this problem. We chose articles to cover a spectrum of software security issues—two case studies on principled methods for building secure systems, development of advanced techniques for creating trust to facilitate software component integration, and source code analysis for software vulnerability detection.

"Correctness by Construction: Developing a Commercial Secure System" by Anthony Hall and Roderick Chapman provides an interesting case study in which good software engineering practices and formal methods are used to demonstrate that we can build practical, secure systems from insecure commercial off-the-shelf components. On a similar theme, "EROS: A Principle-Driven Operating System from the Ground Up" by Jonathan Shapiro and Norm Hardy offers an experience report about building a capabilities-based operating system from the ground up to be verifiably secure.

In "Composing Security-Aware Software," Khaled Md Khan and Jun Han describe their component security characterization framework for composing trust in systems by exposing the components' security properties through active interfaces. Finally, "Improving Security Using Extensible Lightweight Static Analysis" by David Evans and David Larochelle describes a lightweight static analysis tool for software developers to eliminate vulnerabilities in source code prior to releasing software.

We hope you find these articles as enlightening on building secure software systems as we found them enjoyable. Please see the "Further Reading" sidebar for additional resources on this important topic.

## Further Reading

### Books:


*Building Secure Software: How to Avoid Security Problems the Right Way*, by John Viega and Gary E. McGraw, Professional Computing Series, Addison-Wesley, ISBN 0-201-72152-X, Reading, Mass., 2001.

*Computer Related Risks*, by Peter G. Neumann, Addison-Wesley, ISBN 0-201-55805-X, Reading, Mass., 1995.

*Security Engineering: A Guide to Building Dependable Distributed Systems*, by Ross J. Anderson, John Wiley & Sons, ISBN 0-471-13892-6, New York, 2001.

*Security and Privacy for E-Business*, by Anup K. Ghosh, John Wiley & Sons, ISBN 0-471-38421-6, New York, 2001.


### Web resources:

CERT Coordination Center: www.cert.org
Common Vulnerabilities and Exposures: http://cve.mitre.org
BugTraq: www.securityfocus.com/archive/1

## About the Authors


**Anup K. Ghosh** is vice president of research at Cigital and an expert in e-commerce security. He is the author of *E-Commerce Security: Weak Links, Best Defenses* (John Wiley & Sons, 1998) and *Security and Privacy for E-Business* (John Wiley & Sons, 2001). His interests include intrusion detection, mobile-code security, software security certification, malicious software detection/tolerance, assessing the robustness of Win32 COTS software, and software vulnerability analysis. He has served as principal investigator on grants from the US National Security Agency, DARPA, the Air Force Research Laboratory, and NIST's Advanced Technology Program. Contact him at anup.ghosh@computer.org.



**Chuck Howell** is consulting engineer for software assurance in the Information Systems and Technology Division at the MITRE Corporation. The Division focuses on exploring, evaluating, and applying advanced information technologies in critical systems for a wide range of organizations. He is the coauthor of *Solid Software* (Prentice Hall, 2001). Contact him at howell@mitre.org.



**James A. Whittaker** is an associate professor of computer science and director of the Center for Software Engineering Research at the Florida Institute of Technology. His interests involve software development and testing; his security work focuses on testing techniques for assessing software security and the construction of tool to protect against any manner of malicious code. Contact him at jw@se.fit.edu.