Editor: Warren Keuffel ■ wkeuffel@computer.org

An Internal Debate and a Flash of Insight

Anthony Akins

Great Software Debates *by Alan M. Davis, John Wiley & Sons, 2004, ISBN 0-471-67523-7, 271 pp., US$45.00.*

It took me a while to figure this book out. Yes, I read the back cover, preface, and introduction, so I knew what the author intended—but I still couldn't figure it out. I felt there was something else hidden in the words, but I couldn't see it.

Great Software Debates by Alan M. Davis is a collection of essays, many of which were originally published in *IEEE Software* while he was the editor in chief. From the software industry to management to requirements to research, the essays—written from 1992 to 2004—capture the divergent thoughts of one of the discipline's most prolific writers. The earlier essays often end with an epilogue that updates the topics.

In the preface, Alan discusses the book's two types of readers—practitioners who can use the essays as evidence and support to initiate change, and students and teachers who can use the essays as starting points for debate and further exploration and study.

Alan arranged the essays by topic. As I read them, I found myself wishing for a chronological table of contents so that I could see how Alan's opinions and interests changed over time. However, the book's topic-oriented organization made it easy to dive into a single essay or read an entire topic.

Great Software Debates invites the reader in, offering nuggets of insight throughout. It was hard to read just one essay in a sitting. Even when a particular essay didn't catch my interest, I often found that the next would. Of the 38 essays, I have a handful of favorites, including the following three.

"Software Lemmings" eloquently describes the annoying behavior we tend to find in this industry. Too often we follow the leader and assume that if "everyone" is taking a certain path, it must be right. At one time every path seemed straight and true, from the structured path to the object path to the process-maturity path to the language paths and so on. Old paths fade away, and new paths will arise. The question is, will we continue to follow whatever new path comes our way?

"Trial by Firing: Saga of a Rookie Manager" relates Alan's first experiences as a manager. If you've been a manager, you'll recognize the challenges, temptations, and failures that Alan experienced. If you're thinking of such a career move, this essay is well worth reading, along with "Can You Survive Your Management Mistakes?" and "Should He Stay or Should He Go? Advice for a Beleaguered Manager." Too many of us think management is easy, but good management might be the hardest job of all.

ONLINE REVIEWS

- **"Software Engineering Development and Solutions"** by Paolo Donzelli
  A review of *Strategic Software Engineering: An Interdisciplinary Approach* by Fadi P. Deek, James A.M. McHugh, and Osama M. Elijabiri.
- **"When Old Math Protects Your Mailbox"** by Radu State
  A review of *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification* by Jonathan A. Zdziarski.

www.computer.org/software/bookshelf

0740-7459/06/$20.00 © 2006 IEEE

"Requirements Are but a Snapshot in Time" is my favorite of Alan's requirements management essays. Given his reputation as an expert in this area, it should be no surprise that he includes several essays on this topic. He offers no easy answers to this problem but does offer his experience and insight. This article reminds us that we can never afford to stop eliciting requirements—as soon as we do, we're doomed to build a useless system that our customers won't want or need.

At first glance, this book seems like a good fit for a computer science seminar. Each essay ends with "seeds for debate"—questions and challenges that help the reader think about the whole essay, looking at all angles, perspectives, and viewpoints. Yet, as I mentioned earlier, I felt like I was missing the book's real value. As I kept reading, that thought festered in the back of my mind.

I didn't figure it out until I had one of those rare moments of insight. I had read several essays in one sitting and then taken a break to let the ideas work around in my mind. That's when it hit me. *Great Software Debates* is a gift of Alan Davis' experience, insight, wisdom, and humor. The essays' value is greatly increased by being collected in one place. Thanks, Alan.

Anthony Akins is a senior consultant at Valtech Technologies. Contact him at anthony.akins@valtech.com.

How to Avoid Writing Vulnerable Code

Radu State

The Software Vulnerability Guide *by Herbert H. Thompson and Scott G. Chase, Charles River Media, 2005, ISBN 1-58450-358-0, 354 pp., US$49.95.*

An ounce of prevention is worth a pound of cure, but modern medicine is essentially based on diagnosing a disease and then treating the observed symptoms. *The Software Vulnerability Guide* follows the former approach by tackling software security at its roots—the developer's initial programming code.

Herbert H. Thompson and Scott G. Chase cover a broad thematic scope, but they wisely lay out some essential security principles, illustrating them with examples and showing how to fix the induced vulnerabilities.

Generally speaking, software flaws let attackers detour a program's execution and perform actions that are otherwise forbidden to them. Attackers can exploit core security flaws to

- execute privileged programs,
- impersonate another user by getting unauthorized access to his or her credentials,
- crack weak passwords,
- buffer and format string overflows,
- abuse temporary files and system memory,
- tamper with parameters in Web-based applications,
- abuse weak input validation techniques, and
- exploit application-specific vulnerabilities.

Thompson and Chase show typical code examples for each of these potential threats and describe how attackers can exploit these vulnerabilities. The book has relevant visual illustrations, in some cases including a step-by-step screenshot presentation. From a technical viewpoint, the book doesn't go into the greatest possible detail; for instance, if we consider buffer and heap overflow, the book doesn't cover writing the code for an exploit and associated shellcode (but it does abundantly discuss a visual-driven debugging session and an illustration of the stack overwriting). Such an approach is welcome because it will help readers understand the principles and acquire the necessary technical background for more advanced readings.

Software flaws let attackers detour a program's execution and perform actions that are otherwise forbidden to them.

Although the overall outline is very good, some content seems out of place and only marginally relevant to the overall topic. For example, a chapter on spoofing and man-in-the-middle network-level attacks doesn't really match the rest of the book's mainstream software content. However, *The Software Vulnerability Guide* also includes many additional goodies, ranging from an introduction to security tools (including disassemblers, assessment tools, and debuggers) to software for finding and exploiting security flaws in Web-based applications.

The book achieves an excellent balance between technical depth and a high-level overview and more conceptually oriented content. This balance is difficult to attain; few books can both address a highly trained software developer community and interest the larger audience of less-skilled programmers.

The Software Vulnerability Guide can be helpful for many different readers. Seasoned software developers will appreciate a refreshing overview. The book can help less experienced programmers avoid the most common security flaws and pitfalls. I highly recommend the book for graduate or advanced undergraduate students in a software security course because it provides essential knowledge for making better, more secure software.

Radu State is a senior researcher at INRIA. He also teaches a graduate-level computer security class at Henri Poincaré University. Contact him at radu.state@loria.fr.