Paul C. van Oorschot
Associate Editor in Chief

Software Security and Systematizing Knowledge

The job of a university professor is an interesting one. As far as I can tell, there is no generally agreed upon job description. A written description would be a vague approximation of what professors actually do anyway. In general, though, two priority components are always noted: research and teaching. These aspects also are highly complementary, in my experience, and both relate directly to practice in computer security. The teaching aspect brings interaction with bright young students. They take courses in subject areas that they have never studied before and then proceed to ask obvious questions such as, "What exactly is software security?" That one turns out to be not very difficult to answer. Until you try to do so.

I was asked this question back in January, around the same time that Gary McGraw announced he was officially retired from producing monthly Silver Bullet Security podcasts, which he ran continuously from April 2006 through December 2018. (Thank you Gary!) McGraw has played a large role in moving software security from a collection of unpleasant practical issues, ranging from nuisance to crisis (the incidents—not Gary), into a recognized subdiscipline of computer security. Reaching over to my bookshelf, I confirm that McGraw's book *Java Security*[1] begins by explaining the Java security model and what a malicious applet is. His follow-up book, *Securing Java*,[2] includes a technical discussion of code signing (digitally signing Java byte code) and guidelines for mobile code security. Why mention these books? They were instrumental at a critical time, setting us on a path.

In retrospect, without industry's early overstated claims regarding Java security, which drew the interest of a few researchers, the entire field of software security and our collective understanding of it may well have been quite different. We may have never seen *Building Secure Software*,[3] a landmark book that gave many general readers their first detailed treatment of buffer overflows in hardcover. McGraw then teamed with Greg Hoglund for *Exploiting Software: How to Break Code*,[4] which offered a discussion of security attack patterns, and *Exploiting Online Games: Cheating Massively Distributed* Systems.[5] Between these was *Software Security: Building Security In*,[6] which included an insightful annotated bibliography.

In parallel, Michael Howard and David LeBlanc, with a finer focus on helping Microsoft developers, delivered *Writing Secure Code*[7] and *Writing Secure Code for Windows Vista*.[8] They later teamed up with John Viega, the first author of the landmark book I mentioned previously and well known to *IEEE Security and Privacy* readers as a former editor-in-chief. Their *Deadly Sins in Software Security* series[9,10] was aimed squarely at developers. The early books inspired others in software security. A favorite is the superb (and hefty, at more than 1,000 pages) *Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*.[11]

What distinguishes these books? Why do they receive special attention? They were written by experts who had first-hand expertise, cared about real products, and wrote about actual vulnerabilities they had seen. More importantly, these books also began the process of seriously consolidating and sharing knowledge in this area, in a way that makes it available to a wider audience including developers, beyond the narrow field of research specialists. These authors conveyed their experiences, included new ideas and observations, and began the process of organizing what was known. Some would argue that this is what science is about (but that is a separate, and longer, discussion for another time).

Digital Object Identifier 10.1109/MSEC.2019.2904127
Date of publication: 14 May 2019

 Copublished by the IEEE Computer and Reliability Societies

Meanwhile, countless practitioners were dealing with software security issues, many entirely independent of other practitioners, putting out fires on a daily basis, experiencing and rediscovering similar things. On yet another channel, experts revealed details of seemingly impossible attacks in a stream of one-off technical articles that appeared online,[12,13] on security mailing lists,[14–16] and in talks at events such as the Chaos Communication Congress and Black Hat.

Many of these individual articles were rich technical contributions, but they did not attempt to consolidate knowledge. They demonstrated impressive technical ideas (some with more of a focus on defense than others). Some built on the ideas of others, but the contributions were often disjoint. Awareness of buffer overflows existed already in the 1970s, albeit within a far narrower audience. Knowledge had not progressed as much as we might have hoped, so many years after the buffer overflow experience in the 1988 Internet Worm.

This brings us back to teaching. And practice. And moving the industry forward. A driving motivation behind the first software security books in the late 1990s and early 2000s was the lack of organized material to help developers improve the security of (or, equivalently, reduce the vulnerabilities in) the software they wrote, not to mention helping students learn these same things. Large corporations (including one headquartered in Redmond, Washington) whose core business was Internet software recognized that this was critically important. Software security services and consulting firms (old giants and newer firms including Cigital and Coverity) whose mission was to help a wide spectrum of clients in this process recognized it also. Thus, it is not surprising who the authors of these early books were or from where they arose.

In these ways, industry and nonacademic experts pioneered software security, ahead of traditional academic researchers. Those in industry certainly experienced the pain of security problems acutely. After a long gap following the Morris Worm, the early 2000s saw a resurgence in malware with Code Red, Nimda, Sircam, Slammer, Blaster, and the like. By a pleasant coincidence, the academic community got a tremendous research boost around this same time, as the Internet bust resulted in a one-time, massive migration of security expertise from industry to academia, headlined by the demise of AT&T Bell Labs and the woes of the global telecommunications industry.

They were written by experts who had first-hand expertise, cared about real products, and wrote about actual vulnerabilities they had seen.

This migration seeded large universities, the vast majority of which had essentially no prior concentration of security researchers, with a wave of senior global security experts as well as young researchers who found the academic world more enticing than an imploding high-tech world. There was a definite (positive) effect, advancing software security significantly—in our capacity for individual research contributions and our ability to contribute to a collective understanding of the field.

So, what is software security? Perhaps the easiest path to an answer is to ask how it differs from security software. Security software performs security-specific tasks and implements security mechanisms. Antivirus (antimalware) software, intrusion-detection tools, firewalls, and cryptographic toolkits, protocols, and algorithms—these all are security software (products and tools). In contrast, software security focuses on the security of software itself, software whose main functionality is not necessarily security (although it could be). In such generic software, security involves issues such as buffer overruns and related memory safety violations, race conditions, integer vulnerabilities, improperly resolved resource references, and privilege escalation.

Therefore, software security is not about embedded security mechanisms per se, or about network appliances that help in security management, but the properties of everyday software and platforms. It leads to the consideration of software tools, the run-time support and training needed—and to reconsidering the base programming languages used to write code—to reduce and mitigate software vulnerabilities.

A significant fraction of attention continues to go to legacy software. The historical tools of choice (hello, C programming language—our best friend and worst enemy) were designed for an environment very different from today's world. They prioritized efficiency and direct access over security and type safety. The security issues related to browser–server interactions and web security more generally (e.g., cross-site scripting, cross-site request forgery, SQL injection) also form a large subcategory of software security. However, the main point to highlight is that we now have a much richer understanding of software security as a discipline than we did in the late 1990s. A multitude of

tools are available to help address the many types of software vulnerabilities that arise.

We have arrived at this point in our understanding of software security by pooling the expertise and strengths of several subcommunities: software practitioners, independent consultants, and security researchers in both academia and industry. In different ways, each contributes toward the overall advancement of knowledge, and it is through the interaction of these subcommunities that true progress is made. This mixed collection, including the support of technical managers, is also the target audience of *IEEE Security & Privacy*. We look forward to your ongoing input and guidance to ensure that each subcommunity continues to be represented in the articles that appear we publish. ■

References

1. G. McGraw and E. W. Felten, *Java Security: Hostile Applets, Holes, and Antidotes*. New York: Wiley, 1996.
2. G. McGraw and E. W. Felten, *Securing Java: Getting Down to Business with Mobile Code*, 2nd ed. New York: Wiley, Jan. 1999.
3. J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston: Addison-Wesley, 2001.
4. G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. Boston: Addison-Wesley, 2004.
5. G. Hoglund and G. McGraw, *Exploiting Online Games: Cheating Massively Distributed Systems*. Boston: Addison-Wesley, 2007.
6. G. McGraw, *Software Security: Building Security In*. Boston: Addison-Wesley, 2006.
7. M. Howard and D. LeBlanc, *Writing Secure Code*. Redmond, WA: Microsoft Press, Oct. 2002.
8. M. Howard and D. LeBlanc, *Writing Secure Code for Windows Vista*. Redmond, WA: Microsoft Press, 2007.
9. M. Howard, D. LeBlanc, and J. Viega, *19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. New York: McGraw-Hill, 2005.
10. M. Howard, D. LeBlanc, and J. Viega, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. New York: McGraw-Hill, 2009.
11. M. Dowd, J. McDonald, and J. Schuh, *Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Boston: Addison-Wesley, 2006.
12. M. Conover and w00w00 Security Development, "w00w00 on heap overflows," 1999. [Online]. Available: https://www.cs.utexas.edu/~shmat/courses/cs361s/w00w00.txt
13. Scut and team teso, "Exploiting format string vulnerabilities," 2001. [Online]. Available: https://www.win.tue.nl/~aeb/linux/hh/formats-teso.html
14. Solar Designer, Bugtraq mailing list, Aug. 1997.
15. Aleph One, "Smashing the stack for fun and profit," 1996. [Online]. Available: http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf
16. Anonymous, "Once upon a free()," *Phrack Magazine*, 2001. [Online]. Available: http://phrack.org/issues/57/9.html