Book Reviews

RIK FARROW AND MARK LAMOURINE

The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win

Gene Kim, Kevin Behr, and George Spafford
IT Revolution Press 2013; 345 pages
ISBN 978-0988262591

Reviewed by Rik Farrow

I learned of this book while working on the LISA '14 tutorial committee. After reading it, I wondered how I had missed hearing about it previously. And although this is a novel, it also provides a deeper understanding of DevOps, something I hadn't encountered before.

The plot roughly follows that of Eliyahu Goldratt's *The Goal,* which has become a model for systems management ever since it was published in 1984. Bill, the protagonist, is the senior IT guy who has risen to management of the mid-level systems group. Bill is happy where he is and manages his own group well, but that safe harbor disappears in the opening chapter, when his CEO deftly maneuvers Bill into taking the VP of IT position. The former VP has disappeared under a cloud, and Bill quickly finds himself having to deal with one impossible situation after another.

For anyone who has worked in IT, the details of the story will sound familiar: failed releases, the over-ambitious project, a release deadline set by marketing, and an IT department that is not just fragmented, but fractious. Bill gets guidance from a new board member, who takes him on visits to a smoothly functioning factory. Rather than tell Bill what to do, the board member provides hints and leaves Bill to work things out on his own. In real life, you could read other Gene Kim books and get a head start. But Bill progresses through one disastrous release after another, getting a handle on development, quality assurance, security, testing, and release management.

I found the book easy to read and breezed through it. If you usually read novels driven by character development, you will find just traces of that here. The greatest benefit to reading this is getting a visceral, on the ground understanding of how workplace transformation can happen, given the right set of circumstances and personalities. The Phoenix Project is not a textbook, but it still gets across key ideas about controlling the acceptance of new projects, uncovering chokepoints, and how continuous integration actually makes software projects more successful and less expensive.

Penetration Testing: A Hands-On Introduction to Hacking

Georgia Weidman
No Starch Press, 2014; 531 pages
ISBN 978-1-59327-564-8

Reviewed by Rik Farrow

This is the book I wish I had when I was teaching my two-day, hands-on Linux security class. At more than 500 pages long, the book covers a lot more material and many more topics than I could in two days.

After a brief introduction, Weidman spends a long chapter on setting up four VMs: one for the pen testing, and three as targets. Although the pen tester's VM runs Kali, a Linux distro that already includes many tools for security, the author takes the time to explain how to install additional tools that will be used in exercises throughout the book. The target systems also get extra attention, with vulnerable apps getting installed. I like this approach, because it prepares the reader for what's to come, as well as encouraging the reader to do more than just read.

The next couple of chapters are the weakest, but they will need to be there for some readers. I do wonder how many people who can't use basic Linux commands will be successful with pen testing, even when using GUI-based tools like Wireshark, or how showing someone a short shell script or C program will help.

Once past this point, Weidman progresses rapidly, providing a quick overview of Metasploit from the command line. In part two, she guides the reader through vulnerability scanning, port scanning, and network packet capture. Weidman's explanations are clear and accurate, if a bit terse. And although she tells the reader to start up Wireshark as root and "click through the warnings about using Wireshark as root being dangerous," I wished she had explained why: that Wireshark itself can be exploited while parsing packets, and that being root makes any exploit much more useful to the attacker. In a book that teaches about vulnerabilities and exploits, I thought explaining this issue would both help with the pen tester's mind-set as well as act as a warning. I found myself imagining an organization's security team exploiting the pen tester's laptop, something which I know has been done, including by one of the people Weidman lists in her acknowledgments. At least Weidman describes running Kali from within a VM, partially excusing her exclusive use of the root account for all exercises throughout the book.

Weidman does a very nice job of explaining how stack-overflow exploits work, as well as going through examples of how to build these exploits. She does also point out that stack-overflows only work on older OS versions, before data execution prevention (DEP) became the norm for most software. Still, with the use of examples, she walks the reader through how these exploits work, essential knowledge for the person who wants to understand exploitation in the post-DEP and address space layout randomization (ASLR) defensive environment. And understanding how to write exploits forms the basis for modifying existing Metasploit exploits or writing new ones, which she covers in a chapter.

Weidman has developed the Smartphone Pentest Framework herself and covers this in the final chapter. SPF works with Android emulators, whose setup is described in the first chapter, but Weidman uses the framework to explain how attacks outside of the simulated environment should work. I did find myself cringing when she suggested changing the SSH password for the iPhone (alpine is the root password), but for the most part, her writing and exposition are solid.

There are also chapters on exploiting Windows and bypassing antivirus, among other topics.

If you are interested in understanding security from the perspective of the practitioner—that is, a pen tester or hacker—*Penetration Testing* will certainly do more than get you started. For many people I taught over the years, this book will explain more about the tools we used then, and about new tools and techniques.

Understanding Computation

Tom Stuart
O'Reilly Media, 2013. 315 pages.
ISBN 978-1-449-32927-3

Reviewed by Mark Lamourine

In *Understanding Computation*, Stuart sets out to provide something you don't often find in the computing aisle of the retail bookstore chains. Most books in this area are tutorials and references designed to achieve a level of popularity by focusing on the most recent languages and frameworks. Stuart, by contrast, takes on an Honest-To-God Theory of Computation. Although this would typically be an academic book, Stuart has put this one together with the professional computing audience in mind.

Stuart breaks the book into two sections (if you exclude the brief introduction to Ruby—more on that later).

In the first half, Stuart builds up simple computational machines, starting with parsers and finite automata and finishing with the development of a universal Turing machine. He explores the capabilities and limits of each machine and then investigates how to extend the machine to the next level.

Stuart has chosen to express the logic of the machines using Ruby rather than a formal language. He acknowledges that this approach poses some tradeoffs in clarity, but he thinks this is offset by the fact that the reader can actually execute and observe the behavior of the machines he describes. I applaud the attempt to invite the reader to experiment and explore, but I think that he might have made the concepts clearer by presenting them in proper notation as well as in code. This would have given a concise representation that could be compared to the working code. As it is, it can be difficult to separate the topical content from the Ruby code artifacts.

Stuart spends the remainder of the book exploring the capabilities and limitations of the universal Turing machine. Again, he starts with the language of computation, this time the lambda calculus. After producing a working implementation in Ruby, he shows that the lambda calculus is equivalent to a universal Turing machine, as are, in the end, several possible alternative computational models. Again, it would have been clearer to include the operations and explanation of the lambda calculus in traditional notation followed by the translation into Ruby.

Finally, in a chapter entitled "Impossible Programs," Stuart confronts the truly difficult problems of modern computation. In a mere 30 pages he treats the identity of code and data, decidability, and the Halting Problem. Godel's Incompleteness Theorem gets mentioned, but there is no real discussion of its deep implications.

Stuart quotes from and provides a reference to Turing's original paper on computability. In at least half a dozen other places, he makes a passing comment about some other research or information that could have added depth to the discussion. In some of those places, he includes a link to Wikipedia (which I think is a fine place to learn more), but in others he just moves on.

Stuart has done a fine job presenting the content of this theory, but the presentation lacks a sense of the significance and wonder that I find in the idea that my laptop is, conceptually, no more powerful than a Turing machine. Nevertheless, *Understanding Computation* is still the only offering that I've seen aimed at computer professionals, and it will serve that audience well.