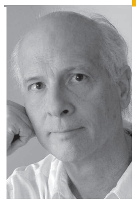
RIK FARROW

musings



rik@usenix.org

SOME GUY FROM QUALCOMM TOLD me, during the USENIX Security program committee party in Vancouver, that he was amazed that I could come up with so many columns. Sometimes I wonder about that too, as I strive not to repeat myself or to travel down already well-worn paths. Just like following a rutted road, it is all too easy to follow the groove.

I've just returned from the USENIX Security conference in Vancouver, the fifteenth such conference, and, as usual, I find myself depressed. Not just because the conference, which itself was a lot of fun, is over, but because not enough has changed.

True, some students showed how they could build a simple device that was able to monitor the keyboard serial cable for possible username/password combinations. The students came up with a scheme that encoded passwords by varying the time between keystrokes by 20 milleseconds, adding some framing, and repeating the same sequence of timings. If those keystrokes travel across a network, even as part of an encrypted SSH connection, they will expose the password.

Isn't anything safe? No, not really, but there certainly are ways that we could make things better. Sitting in on the 2006 USENIX/ACCURATE Electronic Voting Technology Workshop (www.usenix.org/events/evt06), I learned that cryptographic techniques which would make voting more accurate have existed for many years. In his presentation, Josh Benaloh of Microsoft Research said that the hard problems in voting, such as accurately recording votes, accurately counting those votes, and providing the voter with proof that her vote was recorded as cast without revealing the way she voted, are manna to cryptographers. Give them a difficult problem, and they eat it up. So now we have a choice of solutions, not just one. Are we using them? Not in the U.S.

Joseph Lorenzo Hall, a lawyer at UC Berkeley, suggested that we need to add a new category to open source licensing—"open disclosure." "Open disclosure" means that vendors maintain both control of and the license to their software, but openly disclose the code so that it can be checked for correctness. This sounds like a great idea, especially in countries where everything gets done for a profit, from prisons to hospitals, and even voting machines. But there are flies in this ointment, as some voting vendors' code is so ugly that they would likely drop out of the business sooner than reveal their code. As he said this, I heard mumblings in the room from people who had actually had permission to audit DRE (Direct Recording Election) machine code, quietly agreeing with him about the crappiness of the code used in popular voting machines.

Another talk really knocked my socks off (yes, I was wearing shoes). Ka-Ping Yee, also of UC Berkeley, talked about work he had done with David Wagner, Marti Hearst, and Steve Bellovin to create voting software. They had split the voting code into two parts: the set of images that represent the ballots and screens that make up the user interface, and the code that interprets the voter's input. Each image, representing a particular contest for, say, corporate commissioners, can be rendered in advance, tested, approved, and digitally signed. The logic that ties the images together, creating a flow from one screen to the next, can also be verfied through testing. Sounds pretty simple, and it actually is. Instead of the 31,000+ lines of C++ code sitting on top of the Microsoft Foundation Libraries and Windows CE that's found in Diebold's DRE system, Ping and his friends wrote their functional e-voting software in 293 lines of Python and just a couple of libraries. Hmmm, seems like that just might be short enough to audit.

There was much more going on during EVT 2006: read the summaries in this issue of *;login:*. I left knowing that it was definitely possible to create trustworthy voting systems and that Europe, Australia, India, and other countries had successfully worked with e-voting systems, but my own country, the U.S., had decided to stick with systems known to be broken or ones still sealed in secrecy.

Illumination

There were certainly bright notes during the Security conference: for instance, Andy Ozment presented a paper showing that code, at least OpenBSD code, actually has been getting better. He and Stuart Schechter showed that most of the security problems found in OpenBSD came from code inherited when OpenBSD was forked from NetBSD. These bugs continued to be found for many years, while a much smaller amount of new code was found to contain bugs as well. Finally, a ray of hope, and one that might apply to other code bases as well.

But I'm still depressed. Unsurprisingly, no new technique for guaranteeing the security of any computer system appeared. Instead, we are besieged by growing complexity. Sure, you can still strip down a UNIX, Linux, or BSD system to its bare minimum and reduce your risk factor enormously. The version of Linux to be used in the One Laptop per Child (laptop.org) project will be extremely stripped-down, as befits an OS designed for a single hardware target (no extra device drivers) to be used by children, with no system administration needed. There are still some floppy-disk versions of UNIX-like operating systems around, and many more that fit into small (16MB) flash devices. These come close.

But when I run a process listing on a Linux, Mac, or Windows system, I am astounded at the number of processes and dismayed at the number of processes I don't recall enabling or needing. The complexity of desktop systems has grown along with the apparent need to coddle the user. Oh, I guess I should write "improve the user's experience." While it is true that automounting a CD and popping open a file browser or music player is a nice feature, it still represents complexity and provides an attacker with the ability to execute commands as the currently logged-in user. If you wonder if this ever happens, just consider the Sony BMG debacle of late 2005, when it was disclosed that millions of Windows systems had a file-hiding rootkit installed in the name of digital rights management (DRM). Ed Felten of Princeton provided a great synopsis of why many vendors do the wrong thing, while Apple has managed to do the right thing with their DRM. And Felten's student, Alex Halderman, gave us the details of the rootkit.

Please do not think that because you use Linux, BSD, or Mac OS X you are not vulnerable. During Black Hat, a conference occurring simultaneously with Security '06 (too bad), two researchers displayed a wireless hack that affects most Wi-Fi devices based on Atheros chips and demonstrated it on Mac OS X Tiger—just because of the perceived security of that OS. No OS is immune to faults, but some systems will have fewer faults than others.

Lineup

In this issue, we lead off with Steve Johnson. In the June '06 issue of *;login:* I mentioned that Steve had found some surprising performance results when he experimented with data locality. Keeping often-referenced elements close together, in structures, has become a mantra in modern programming (see Spinellis in the April 2006 *;login:*), but Steve clearly shows that this may not be the best strategy. Programmers (and system designers), take note!

Mike Howard sounds off next with a response to Luke Kanies's article (*;login:*, April 2006) about Ruby. Mike considers Python nearly as object-oriented (completely so in v2) as Ruby, with many of the same features as Ruby but more maturity.

In the Sysadmin section, Mark Burgess continues his series on configuration management. He is followed by Brad Knowles, who discusses in detail the state of NTP, providing excellent advice about the appropriate use of NTP and stratum one and two timeservers.

The Technology section brings something for which I have been searching: a discussion of CPU and system technology designed to deal with the large difference between CPU and memory speed. Richard McDougall and James Laudon write about Sun's new T1 CPU architecture. In a computing world dominated by Intel and AMD, Sun has taken a very different tack, reverting to an earlier processor design, then building an eight-core, multi-threaded system. The T1 architecture provides some very significant throughput gains in applications that already have many threads, such as Apache and Oracle, while using much less power (and producing less waste heat) than their powerful competitors' chips. People whose applications match this processor's strengths owe it to themselves (and their energy budgets) to take a close look at this technology.

Timo Sivonen then explores two techniques for using encrypted file systems within FreeBSD. Timo explains how he tried two GEOM encryption facilities, GBDE and GELI, and compared their performance, including the use of different encryption algorithms.

In the Network section, Mike Freedman writes about OASIS, a system for automating server selection. Like two articles that appeared in the June '06 issue of *;login:*, this article is based on a paper that appeared at NSDI ’06. The OASIS system provides ways that clients can be directed to the most appropriate server. Most systems for choosing the best of a set of replicated servers choose the closest server, but the OASIS algorithm also takes into account the current load reported by each server.

I wrote the next article in response to a request from Teus Hagen of NLnet. Teus thinks that the world needs a project to create a low-cost networking infrastructure. As I dug into this topic, I could see that while there are some projects dancing around the edges of this issue, what Teus has in mind goes much further. This article explores some issues in wireless technology, using RoofNet as an example, then ends with Teus’s wish list for this new technology.

David Blank-Edelman has written the second half of column about tie(), outdoing himself (as usual), while Robert Haskins takes a look at the world of anti-spam solutions. Heison Chak considers how echo arises in VoIP. Robert Ferrell is back with another /dev/random column, to be taken not seriously but certainly thoughtfully. After an excellent selection of book reviews, two articles about the standards process, and USENIX Notes, this issue ends with an array of conference reports: 2006 USENIX Annual Tech, SRUTI ’06, and EVT ’06.

The mention of standards reminds me of something I wanted to include in this Musings. Nick Stoughton’s description of work on the ISO-C committee sent me to an article by Dennis Ritchie on the birth and early life of the C programming language. I enjoyed reading Nick’s article, but I equally appreciated Dennis’s viewpoint on the development of C (and reading about Steve Johnson’s part in this). You can find this article at http://cm.bell-labs.com/cm/cs/who/dmr/chist.html.

I will confess that I see some features of the C language very differently from that of one of its creators. I heard this echoed during USENIX Security, where one panelist described C as “the best macro-assembler ever written.” My own view of C is similarly colored, as I found it wonderfully close to the assembly I was using when I first learned C. But I also like to call C the programming language for people who write operating systems, and I hope that the many programmers who don’t write operating systems will consider writing in strongly typed languages that have bounded arrays and don’t allow manipulation of pointers. The computing world would be a much safer place if they did so.