The Virtue of Paranoia

For the past three-and-a-half years, Kode Vicious has guided many a befuddled programmer toward clarity and understanding. We hope you value his truths from the trenches, and will continue reading the column and sending him your queries as *Queue* transitions from print to digital. As our digital subscribers already know, e-mailing him will be as simple as clicking on kv@acmqueue.com. He hopes to hear from you soon.

Dear KV,

I just joined a company that massages large amounts of data into an internal format for its own applications to work on. Although the data is backed up regularly, I have noticed that access to this data, which has accumulated to be several petabytes in size, is not particularly well secured. There is no encryption, and although the data is not easily reachable from the Internet, everyone at the company has direct access to the volumes, both physically and electronically, all the time. Our data center is not particularly well protected either, with just two locked office doors between the outside world and the machines inside.

I have tried to convince my management that we need to do more to protect the data, but they argue that once the data is massaged into an internal format, it's not really of use to anyone else; and that as long as we have backups, and therefore would not suffer an interruption should a theft occur, we are adequately secured. How do I get them to see the value of the data that we have and to do more to protect it?

Petabytes of Paranoia

Dear Peta,

If it's any consolation to you, and I know that people write to KV looking to be consoled, you are not alone in

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

your plight. Many people undervalue their data, believing that it can be of little use to anyone else.

Although more people are coming to understand the risk of leaking databases of personal information, such as credit cards and medical records, many other types of data remain unprotected.

Another way to think about the value of data is to ask, "How much damage could be done to me, or my company, should another party get this data?" The competitive advantage that a company has based on its data is, in most cases, the best way to value that data.

Is the data worth more as it ages? Or is it worth less? If data is worth less with age, then the best way to protect it, if the law does not require that it be kept, is to throw it away. No, I do not mean dragging it all to the little trash can or recycle bin on your desktop; I mean securely disposing of the data. Some companies will destroy your disks for you, if you're feeling particularly paranoid. In most cases, however, using a secure erase command, such as `rm -P` on FreeBSD, is sufficient. Again, it's all about how much that data is worth should it be found by others.

One other way of scaring your bosses into securing the data is to perform a simple search for recent cases of physical data theft. Many companies have been targeted and successfully attacked in this way, including ones that stored their data in secure data centers. Armed robberies of data do happen.

I would like to say that it's hard to imagine people not understanding the value of their data in this day and age, but unfortunately it is all too easy to imagine. Perhaps what your bosses lack isn't knowledge but imagination.

KV

Dear KV,

My group has been maintaining an old CMS (content management system) for several years, and we think it's time for an upgrade. The system is used by a bunch of text monkeys to manage the pages on our Web site. Since we're a Web company, this is a pretty important system. The code was written in-house, but the original team has

left and the system has been in maintenance mode for five years. There have been several attempts to replace the original system, but each of these has failed, usually because some savior comes in at the last minute and addresses an issue or bug in the original system.

I've been asked to evaluate new CMSes. Quite a few are available, including both open and closed source systems. One problem that I have found is that many of these systems seem to be written in very high-level languages and run considerably slower than the code we already have. I can't imagine recommending a system that is newer but also slower than what we have now. Why would these new systems be so slow, and what do you do when you're stuck between a rock and a hard place like this?

Managed Contents

Dear MC,

When I'm stuck between a rock and a hard place, I tend to lean on the rock, as it relaxes my back, but that's probably not the advice you need.

Although you, perhaps wisely, do not list the systems that you're evaluating, I suspect they all have one thing in common: each one is a framework written within a framework. What do I mean by this? Am I just ranting again? Has KV lost his mind? Who would be crazy enough to build a recursive framework?

While, yes, I am ranting again, and yes, I clearly have lost my mind, there is, somewhere in these lines, a point to all this. The basic problem that you are seeing is not a result of newer systems just having more features than your old in-house system had, but a symptom of a pervasive sickness in the programming world.

Many programmers have gone so far up the software stack, far from the actual hardware, to such high levels of abstraction that they have forgotten how computers work. Some programmers have never really learned how computers work at the machine level, so they make decisions that inevitably hamper system performance.

Why would understanding how a computer works matter? I have found that people who have had close interactions with the lower levels of software and hardware invariably have a better understanding of the impact of their choices on the performance of systems, as well as other aspects such as debugging. It is becoming harder for me to avoid a cliché such as "Back in the old days we didn't have fancy...." Whatever the fancy thing is, we didn't have it. But, really, this need have nothing to do with age; it has to do with changes in how people have programmed over the years.

In the past 30 or so years people have continued to believe that "if only we could make programming easy

and more natural, projects would finish on time." The fact is, many of the efforts to make programming easier and more natural have had good benefits, such as higher-level languages, modularity, and some aspects of object-oriented programming. Unfortunately, as in any endeavor that includes people, some snake oil always manages to sneak in. Many so-called improvements have served only to give the appearance of advancement but then quickly evaporated, leaving behind a bitter aftertaste.

One current fad in this direction is to take domain-specific languages and attempt to build general frameworks or platforms with them. A language that is intended to do something specific, such as lay out Web forms, can be made to do something more general, but there is a price to pay. In computer science we usually dance with the devil at the crossroads of performance, and that is where most sacrifices are made.

A lesson that seems not to have been learned is that just because you *can* do something doesn't mean that you *should* do something. Writing an operating system in PHP is certainly possible, but should you? Probably not. Nor should you write a CMS in assembler. You can implement a framework in a framework, but you're going to pay a heavy cost, because frameworks are meant to make writing a specific program easier, but usually are not written to write another framework. If the original author wanted to write a second framework, he or she would have either written the first one with more features or created a second, domain-specific framework for another domain.

Good programmers know that every extra layer or abstraction lowers the overall performance of the system, so they add only the extra layers or abstractions that are absolutely necessary to get the job done.

In sum, it pays to look under the hood and see if the appropriate technology is being used, and if the people who built the system really understand what happens down below, or if they're just people who know how to make a cool demo.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who has made San Francisco his home since 1990.

© 2008 ACM 1542-7730/08/0500 \$5.00