KV the Konqueror

It's been a couple of months, and Kode Vicious has finally returned from his summer vacation. We asked him about his travels and the only response we got was this: "The South Pole during winter ain't all it's cracked up to be!" Fortunately, he made it back in one piece and is embracing the (Northern hemisphere's) late summer balminess with a fresh installment of koding kwestions. This month, KV follows up on a security question from a previous column and then revisits one of koding's most divisive issues: language choice. Welcome back!

Dear KV,

Suppose I'm a customer of Sincere-and-Authentic's ("Kode Vicious Battles On," April 2005:15-17), and suppose the sysadmin at my ISP is an unscrupulous, albeit music-loving, geek. He figured out that I have an account with Sincere-and-Authentic. He put in a filter in the access router to log all packets belonging to a session between me and S&A. He would later mine the logs and retrieve the music—without paying for it.

I know this is a far-fetched scenario, but if S&A wants his business secured as watertight as possible, shouldn't he be contemplating addressing it, too? Yes, of course, S&A will have to weigh the risk against the cost of mitigating it, and he may well decide to live with the risk. But I think your correspondent's suggestion is at least worthy of a summary debate—not something that should draw disgusted looks!

There is, in fact, another advantage to encrypting the payload, assuming that IPsec (Internet Protocol security) isn't being used: decryption will require special clients, and that will protect S&A that much more against the theft of merchandise.

Balancing is the Best Defense

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.

Dear Balancing,

Thank you for reading my column in the April 2005 issue of *Queue*. It's nice to know that someone is

paying attention. Of course, if you had been paying closer attention, you would have noticed that S&A said, "In the design meeting about this I suggested we just encrypt all the connections from the users to the Web service because that would provide the most protection for them and us." That phrase, "just encrypt all the connections," is where the problem lies.

Your scenario is not so far-fetched, but S&A's suggestion of encrypting all the connections would not address the problem. Once users have gotten the music without their evil ISP's sniffing it, they would still be able to redistribute the music themselves. Or, the evil network admin would sign up for the service and simply split the cost with, say, 10 of his music-loving friends, thereby getting the goods at a hefty discount. What S&A really needs is what is now called digital rights management. It's called this because for some reason we let the lawyers and the marketing people into the industry instead of doing with them what was suggested in Shakespeare's *Henry VI*.

What S&A failed to realize was that the biggest risk of revenue loss was not in the network, where only a small percentage of people can play tricks as your ISP network administrator can, but at the distribution and reception points of the music. Someone who works for you walking off with your valuable information is far more likely than someone trying to sniff packets from the network. Since computers can make perfect copies of data (after all, that's how we designed these things in the first place), it is the data itself that must be protected, from one end of the system to the other, in order to keep from losing revenue.

All too often, people do not consider the end-to-end design of their systems and instead try to fix just one part.

KV

Dear KV,

Since there was some debate in my company over the following issue, I'm curious to see what you believe: put-

ting aside performance issues (which I think are relatively minor on modern PCs), when would you recommend using C++ for development, and when would you recommend C? Do you think it is always better to use C++?

My feeling is that unless your application is inherently object oriented (e.g., user interfaces), C++ will tend to make the implementation worse instead of making it better (e.g., constructors and operators doing funny unexpected things; C++ experts trying to “use their expertise” and writing C++ code that is very efficient but extremely hard to read and not portable; huge portability—and performance—issues when using templates; incomprehensible compiler/linker error messages; etc., etc.). I also think that although people can write bad C code (gotos out of macros was a nice one), typically people can write *awful* C++ code. Where do you stand on this dispute?

Wondering How Much + There is in ++

Dear Wondering,

Choosing a language is something I’ve addressed before in other letters, but the C vs. C++ debate has raged as long as the two languages have been in existence, and, really, it’s getting a bit tiring. I mean, we all know that assembler is the language that all red-blooded programmers use! Oh, no, wait, that’s not it.

I’m glad you ask this question, though, because it gives me license to rant about it—and to dispel a few myths.

The first, and most obvious, myth in your letter is that user interfaces are inherently object oriented. Although many introductory textbooks on object-oriented programming have user interfaces as their examples, this has a lot more to do with the fact that humans like pretty pictures. It is far easier to make a point graphically than with text. I have worked on object-oriented device drivers, which are about as far as you’ll ever get from a user interface.

Another myth that your letter could promulgate is that C is not an object-oriented language. A good example of object-oriented software in C is the vnode filesystem interface in BSD Unix and other operating systems. So, if you want to write a piece of object-oriented software, you can certainly do it in C or C++, or assembler for that matter.

One final myth, which was actually dispelled by Donn Seeley in “How Not To Write Fortran in Any Language” (*ACM Queue*, December/January 2004-2005:58-65), is that C++ leads to less understandable code than C. Over the past 20 years I have seen C code that was spaghetti and C++ code that was a joy to work on, and vice versa.

So, after all that myth bashing, what are we left with?

Well, the things that are truly important in picking a language are:

- What language is most of the team experienced in? If you’re working with a team and six out of eight of them are well versed in C but only two know C++, then you’re putting your project, and job, at risk in picking C++. Perhaps the two C++ koders can teach the C folks enough C++ to be effective but it’s unlikely. To estimate the amount of work necessary for a task, you have to understand your tools. If you don’t normally use a nail gun, then you’re likely to take someone’s toe off with it. Losing toes is bad, as you need them for balance.
- Does the application require any of the features of the language you’re using? C and C++ are a lot alike as languages (i.e., in syntax), but they have different libraries of functions and different ways of working that may or may not be relevant to your application. Often realtime constraints require the use of C because of the control it can provide over the data types. If type safety is of paramount importance, then C++ is a better choice because that is a native part of the language that is not present in C.
- Does the application require services from other applications or libraries that are hard to use or debug from one or the other language? Creating shim layers between your code and the libraries you depend on is just another way of adding useless, and probably buggy, code to your system. Shim layers should be avoided like in-laws. They’re OK to talk about, and you might consider keeping them around for a week, but after that, out they go as so much excess, noisy baggage.

There are lots of other reasons to choose one language over another, but I suspect that the three listed here should be enough for you and your team to come to some agreement. You’ll notice that none of them has to do with how easy it is to understand templates or how hard it is to debug with exceptions.

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.

© 2005 ACM 1542-7730/05/0900 \$5.00