The Doctor is In

KV is back on duty and ready to treat another coding illness: bad APIs. This is one of the most widespread pathologies affecting, and sometimes *infecting*, us all. But whether we write APIs or simply use APIs (or both), we would all do well to read on and heed the vicious one's advice. And as always, your ongoing code-related questions are welcomed and appreciated: kv@acmqueue.com.

Dear Kode Vicious,

I've been reading your rants for a few months now and was hoping you could read one of mine. It's a pretty simple rant, actually: it's just that I'm tired of hearing about buffer overflows and don't understand why anyone in his or her right mind still uses `strcpy()`. Why does such an unsafe routine continue to exist at all? Why not just remove the thing from the library and force people to migrate their code? Another thing I wonder is, how did such an API come to exist in the first place?

Yours for Better APIs

Dear YBAPI,

Yes, it's true, some APIs just seem to be obtuse or written to trip you up. Usually this is not due to evil intent on the part of the koder. As my grandmother used to say, "Never attribute to malice that which can be adequately explained by stupidity." Oh, wait, no, my grandmother said, "If you can't say something nice about someone, don't say anything at all." I have given only brief attention to both of these pieces of advice throughout my life, but my grandmother was a wise woman. The fact is that you can't even blame stupidity most of the time; you most often have to blame the inability of people to be omniscient.

You see, way back in the mists of time, computers

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.

weren't networked and were programmed by a small group of dedicated professionals using a well-constructed set of tools

and libraries. These professionals understood their tools intimately and didn't really think about people attacking their computer programs because many of them worked in research labs, and because most of their programs didn't handle money. Certainly some of these people thought about security, but not in the way one would have to think about it after hundreds of millions of people gained access to computers and the Internet. Before we hooked everything to the Internet, life was good—programmers laughed and played all day, while dreaming of larger disk drives and dynamic RAM. At least, that's the story as I've heard it. So, at the time that `strcpy()` was written, most programmers thought only about their own mistakes, as opposed to someone trying to take over their computers via the network and a buffer overflow attack.

As you said, though, the buffer overflow attack has been discussed to death, and perhaps we ought to think about what makes `strcpy()` such a problematic API instead of hammering on buffer overflows. After all, people are still building APIs that are insecure and poorly thought out, and perhaps we should shove them, if not into the sea, then in the right direction.

Part of the problem comes from the definition of the string itself. A string is just a pointer to a NULL-terminated set of bytes. Let's think about some things we would need to know before passing this hunk of memory around to other APIs. One important question is, "How big is it?" Yes, a bit off-color, but in this case, size actually does matter. If you are on the receiving end of a string, and you don't know how long it is, there really is no way to handle it safely. You have to scan the entire thing until you find the terminating NULL, and even when you do, it might be the wrong one.

A second problem with strings really has to do with how memory is allocated and controlled in programs. Pointers to memory tell you only where the memory starts, not how much you're really supposed to use. Since

it is more efficient to manage memory in terms of groups of bytes, which the operating system calls *pages*, your program is not going to get a clear signal if it accidentally writes past the space you thought was allocated to it. There is no way for the program to know, without the use of special tools and libraries, when it has gone too far. Of course, the special tools and libraries slow your program down so you can't use them all the time, and even when you do, you have to design sufficient tests to see if your code has any holes in it.

And so now we come to `strcpy()`, which for those who may not have ever seen this routine, looks like this:

```
char *strcpy(char *destination, char *source)
```

and which is supposed to copy bytes from source to destination, including the terminating NULL byte, so that when the routine returns, you have a copy of source pointed to by destination. This API has several problems:

- Nowhere in the API is there any indication as to the length of the strings involved. The routine is trusting the programmer to pass a pointer in destination with enough space for all the bytes in source.
- The API has no way to communicate an error. The only way you receive an error from this routine is when the program crashes. At least if the API had a way to return an error, we would have a chance of finding out that something was wrong.
- The API's return value is something you already know! Yes, that's right, you get back a pointer to destination,

You get back a pointer to destination, which presumably you passed to the API in the first place. The logic for this escapes me.

which, presumably, you passed to the API in the first place. The logic for this escapes me.

- The API does not check its arguments for NULL, or to see whether the two areas of memory might overlap, and the documentation does not warn you of this. Although we koders are supposedly famous for not reading documentation, you can't really blame us when it was not written in the first place!

So, let's abstract the bad qualities a bit and try to state them more clearly. First of all, there is no way for the API to validate its arguments. Not validating arguments leads

to errors. Errors are bad. Secondly, there is no clear way to communicate an error status. Errors happen; they should be checked for and returned. The programmer who does not check for errors is a bad person and will suffer eternal debugging sessions forever after, amen. Thirdly, the arguments and return values are confusing. Why return something you already passed into the routine when you don't need to? Lastly, the documentation does not warn us about any of the possible boundary conditions and what we might expect if they occur.

I could, of course, go on and on and find horrific APIs that make `strcpy()` look like a walk in the park with Aunt Rose, but I'm limited to 1,200 or so words. So, what do good APIs look like or what *should* they look like? Well in KV's highly biased opinion, a good API has several attributes:

- The arguments are verifiable by the caller and the callee. For example, if you pass around a pointer to memory and also pass an argument, that indicates how much of that memory is valid.
- Every API must be able to indicate an error, and how that error is indicated needs to be well documented. Set `errno` if you like, but let the callers know that they have to check it and what to check for. I prefer APIs that return an error indication simply because then I don't have to worry about checking the `errno`, and what I really hate are functions that silently fail.
- Do not create overly complex APIs. This is not the case with `strcpy()`, but there are plenty of examples of libraries that seem to revel in allowing you to pass 12 arguments to a function, and if you get one wrong, you have a hell of a time figuring out which argument is the broken one. Or, perhaps, the function has all these options you don't need, and that no one ever needed, but that wound up in there anyway, "just in case." Don't try to anticipate too much. Simpler is often better. Just a few simple rules to code by, from,

KV

CODE 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 1542-7730/05/1100 \$5.00