MUST and MUST NOT

ON WRITING DOCUMENTATION

Dear KV,

I've joined a small security startup and have been tasked with writing up our internal security processes. The problem is that I'm not a writer, I'm a software engineer, and whenever I start trying to write about our processes, I either stare at a blank screen until I get frustrated and look away to do something else, or I just wind up writing a lot of sentences that later don't seem to make a lot of sense. I am sure there must be a template that I can work from to get all these things in my head written down in a useful way, but I'm not sure where to look. For example, I want a way to describe to people what they should and shouldn't do with our software and how it must be used so that it provides the security properties they expect. What I see when I try to write about this is a tangled web of spaghetti text.

Tangled

Dear Tangled,

Normally I would reply that the only way to get a good spate of writing done is to go on a three-day bender, and then before sobering up, sit at the keyboard and pour your heart and soul into your text buffer, save your work, and go on another bender before reading what you wrote. It may not work, but the benders ought to be a lot of fun.

In fact, what I'm going to do is recommend to you a more than 20-year-old document, RFC 2119. KV has mentioned

A long list of **MUSTs** and **MUST NOTs** will be tedious and boring and lose a reader's attention.

RFC (Requests for Comments) before; this is the set of documents going back to the early 1970s in which the Internet protocols and many others are described. For those who are unfamiliar with these documents, they always specify which parts of a protocol are required or optional using a small number of key words:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" [cite RFC 2119]

The meanings of these words are codified in two pages in ASCII, a now-ancient standard for textual communication. These key words are CAPITALIZED as their only form of emphasis. It turns out that it's not necessary to have fancy formatting in order to communicate clearly; in fact, fancy formatting often distracts from the message you are trying to get across.

No, I'm not merely suggesting you use language like this; I believe you **MUST** use these terms as written and then cite the RFC. Getting a group of people to understand your meaning by citing, and perhaps beating them with, a well-known and well-written document can save you a lot of time and trouble. The longer a document is, the more there is to argue over and the more nits there are to pick. Reducing nitpicking saves a lot of time.

A word of caution when using these terms in a security document as you plan to do: The words must be used carefully and for greatest effect. A long list of **MUSTs** and **MUST NOTs** will be tedious and boring and lose a reader's

attention. Inattentive readers make mistakes, and in this case, they will be security mistakes, which are the kinds of mistakes your document is trying to help them avoid. Let me share one more paragraph from the RFC:

These terms are frequently used to specify behavior with security implications. The effects on security of not implementing a MUST or SHOULD, or doing something the specification says MUST NOT or SHOULD NOT be done may be very subtle. Document authors should take the time to elaborate the security implications of not following recommendations or requirements as most implementors will not have had the benefit of the experience and discussion that produced the specification.

What this paragraph says is, “Explain yourself!” Pronouncements without background or explanatory material are useless to those who are not also deeply steeped in the art and science of computer security or security in general. It takes a particular bent of mind to think like an attacker and a defender all at once, and most people are incapable of doing this; so, if you want the people reading the document to follow your guidance, then you must take them on a journey from ignorance to knowledge. Only then can you expect them to properly implement your guidance, in both familiar and—especially—unfamiliar situations.

KV

Related articles

➡ Microsoft's Protocol Documentation Program: Interoperability Testing at Scale
A discussion with Nico Kicillof, Wolfgang Grieskamp, and Bob Binder
<https://queue.acm.org/detail.cfm?id=1996412>

➡ The Next Big Thing
Kode Vicious
<https://queue.acm.org/detail.cfm?id=1317398>

➡ The Robustness Principle Reconsidered
Seeking a middle ground
Eric Allman, Sendmail
<https://queue.acm.org/detail.cfm?id=1999945>

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. Neville-Neil is the coauthor with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System (second edition)*. He is an avid bicyclist and traveler who currently lives in New York City.

Copyright © 2019 held by owner/author. Publication rights licensed to ACM.