This document details a few steps and decisions taken to ensure vsftpd is free of common implementation flaws.

Tackling the buffer overflow

Probably the most common implementation flaw causing security problems is the buffer overflow. Buffer overflows come in many shapes and sizes - overflows onto the stack, overflows off the end of dynamically malloc()'ed areas, overflows into static data areas. They range from easy to spot (where a user can put an arbitrary length string into a fixed size buffer), to very difficult to spot - buffer size miscalculations or single byte overflows. Or convoluted code where the buffer's definition and various usages are far apart.

The problem is that people insist on replicating buffer size handling code and buffer size security checks many times (or, of course, they omit size checks altogther). It is little surprise, then, that sometimes errors creep in to the checks.

The correct solution is to hide the buffer handling code behind an API. All buffer allocating, copying, size calculations, extending, etc. are done by a single piece of generic code. The size security checks need to be written once. You can concentrate on getting this one instance of code correct.

From the client's point of view, they are no longer dealing with a buffer. The buffer is encapsulated within the buffer API. All modifications to the buffer safely go through the API. If this sounds familiar, it is because what vsftpd implements is very similar to a C++ string class. You can do OO programming in C too, you know ;-)

A key point of having the buffer API in place is that it is MORE DIFFICULT to abuse the API than it is to use it properly. Try and create a buffer memory corruption or overflow scenario using just the buffer API.

Unfortunately, secure string/buffer usage through a common API has not caught on much, despite the benefits it brings. Is it under publicised as a solution? Or do people have too much sentimental attachment to strcpy(), strlen(), malloc(), strcat() etc? Of notable exception, it is my understanding that at least the rather secure qmail program uses secure buffer handling, and I'd expect that to extend to all Dan Bernstein software. (Let me know of other good examples).