10 Doing Right by Neighbor O'Hara

by Andreas Bogk
Knight in the Grand Recursive Order of the Knights of the Lambda Calculus
Priest in the House of the Apostles of Eris

What good is a pulpit that can't be occasionally shared with a neighborly itinerant preacher? In this fine sermon, Sir Andreas warns us of the heresy that "input sanitation" will somehow protect you from injection attacks, no matter what comes next for the inputs you've "sanitized"—and vouchsafes the true prophecy of parsing and unparsing working together, keeping your inputs and outputs valid, both coming and going. —PML

Brothers, Sisters, and Variations Thereupon!

Let me introduce you to a good neighbor. Her name is *O'Hara* and she was born on *January 1st in the year 1970* in Dublin. She's made quite an impressive career, and now lives in a nice house in *Scunthorpe, UK*, working remotely for *AT&T*.

I ask you, neighbors: would you deny our neighbor O'Hara in the name of SQL injection prevention? Or would you deny her date of birth, just because you happen to represent it as zero in your verification routine? Would you deny her place of work, as abominable as it might be? Or would you even deny her place of living, just because it contains a sequence of letters some might find offensive?

You say no, and of course you'd say no! As her name and date of birth and employer and place of residence, they are all valid inputs. And thou shalt not reject any valid input; that truly would not be neighborly!

But wasn't input filtering a.k.a. "sanitization" the right thing to do? Don't characters like `'` and `&` wreak unholy havoc upon your backend SQL interpreter or your XHTML generator?

So where did we go wrong by the neighbor O'Hara?

There is a false prophesy making the rounds that you can protect against undesirable injection into your system by "input sanitization," no matter where your "sanitized" inputs go from there, and no matter how they then get interpreted or rendered. This "sanitization" is a heathen fetish, neighbors, and the whole thing is dangerous foolery that we need to drive out of the temple of proper input-handling.

Indeed, is the apostrophe character so inherently dirty and evil, that we need to "sanitize" them out? Why, then, are we using this evil character at all? Is the number 0 evil and unclean, no matter what, despite historians of mathematics raving about its invention? Are certain sounds unspeakable, regardless of where and when one may speak them?

No, no, and no—for all bytes are created equal, and their interpretation depends solely on the context they are interpreted in. As any miracle cure, this snake oil of "sanitization" claims a grain of truth, but entirely misses its point. No byte is inherently "dirty" so as to be "sanitized" as such—but context and interpretation happeneth to them all, and unless you know what these context and the interpretations are, your "sanitization" is useless, nay, harmful and unneighborly to O'Hara.

The point is, neighbors, that at the input time you cannot possibly know the context of the output. Your input sanitation scheme might work to protect your backend for now—and then a developer comes and adds an LDAP backend, and another comes and inserts data into a JavaScript literal in your web page template. Then another comes and adds an additional output encoding layer for your input—and what looked safe to you at the outset crumbles to dust.

The ancient prophets of LISP knew that, for they fully specified both what their machine read, and what it printed, in the holy *REPL*, the Read-Eval-Print Loop. The *P* is just as important as the *R* or even the *E*—for without it everything falls to the ground in the messy heaps that bring about XSS, memory corruption, and packet-in-packet. *Pretty-printing* may sound quaint, a matter unnecessary for "real programmers," but it is in fact deep and subtle—it is *unparsing*, which produces the representation of parsed data suitable for the next context it is consumed in. They knew to specify it precisely, and so should you.

So what does the true prophecy look like? Verily sanitize your input—to the validity expectations you have of it. Yet be clear what this really means, and *treat the output with as much care as you treat the input*—because the output is a language too, and must be produced according to its own grammar, just as validating to the input grammar is the only hope of keeping your handler from pwnage.

Sanity in input is important in structured data. When you expect XML, you shall verify it is XML. When you expect XML with a Schema, also verify the schema. Expecting JSON? Make sure you got handed valid JSON. Use a parser with the appropriate power, as LangSec commands. Yet, if your program were to produce even a single byte of output, ask—what is the context of that output? What is the expected grammar? For verily you cannot know it from just the input specification.

Any string of characters is likely to be a valid name. There is nothing you should really do for "sanitation," except making sure the character encoding is valid. If your neighbor is called O'Hara, or Tørsby, or Åke, make sure you can handle this input—*but also make sure you have the output covered!*

This is the true meaning of the words of prophets: input validation, however useful, cannot not prevent injection attacks, the same way washing your hands will not prevent breaking your leg. Your output is a language too, and unless you generate it in full understanding of what it is—that is, unparse your data to the proper specification of whatever code consumes it—that code is pwned.

Parsing and unparsing are like unto the two wings of the dove. Neglect one, and you will not get you an olive branch of safety—nay, it will never even leave your ark, but will flap uselessly about. Do not hobble it, neighbors, but let it fly true—doing right by neighbors like O'Hara both coming and going!

EOL, EOF, and EOT!

"CHOISA"

CEYLON TEA

Pure Rich Fragrant

Packed in Parchment-lined
One Pound and Half-pound Canisters

1-lb. Canisters, 60 cents
1-2 lb. Canisters, 35 cents

WE INVITE COMPARISON WITH OTHER TEAS
OF THE SAME OR HIGHER PRICE

S. S. PIERCE CO.

Tremont and Beacon Streets
Copley Square . . . . } BOSTON Coolidge Corner } BROOKLINE