Walls and Gates

Why are computers so insecure? Can we come up with better designs? I think so; let's look at how.

Fundamentally, our security mechanisms are built around walls and gates. The walls separate entities that have different privileges; the gates allow restricted communications through the walls.

This terminology isn't new, of course. We all know what a firewall is, but the usage is much older than that; going way back in computer security history, the Multics system used "gates" to pass control between different protection rings. Even the observation that our security mechanisms are based on these two concepts isn't novel; Wulf and Jones noted several years ago that the cybersecurity paradigms we rely on today were essentially unchanged since 1974.[1] My only disagreement with that claim is that I'd have said 1960 or thereabouts. The question, though, is what to do about it.

Before we move on to that, we need to understand why walls and gates don't protect us adequately. The answer is the word "restricted" in the definition from the first paragraph. Restriction implies a policy; that in turn implies both a specification and an implementation of that policy. Either or both, as it turns out, can be flawed. That in turn generally leads to complete system penetration from one bug, a problem I pointed out in this space some years ago.[2] In order to make progress in security, we need a design that strengthens the weak point: the gates. An immediate corollary of this is that building better walls—say, relying on virtual machines instead of separate userids—is not likely to help much; our problem isn't with the walls. (Think: how frequently do you see failures of security primitives, such as access control? Compare that with how frequently network daemons or privileged programs fall over when presented with unusual input.)

This realization, that our security problems are due to errors in the specification or implementation of policies, leads immediately to another conclusion: the "gatekeeper", if you will, needs to be as simple as possible. *All* correctness techniques, from testing to code inspection to formal proofs, are far more tractable and reliable when the programs are simpler. Complexity should live at a single privilege level, isolated by strong walls and simple gates from other privilege levels. When we don't follow that principle, security failures become more likely. One obvious example is the increasing futility of firewalling: the policies implemented by a modern firewall are too complex to be correct. Too many protocols are allowed through, and the content types they're allowed to pass—HTML with JavaScript, PDF files with their own active content, and more—are themselves too complex to be

[1] William Wulf and Anita Jones, "Reflections on Cybersecurity", *Science* 326:5955, pp 943-944, 2009.
[2] Steven M. Bellovin. On the brittleness of software and the infeasibility of security metrics. *IEEE Security & Privacy*, 4(4), July-August 2006.

filtered. (The other reason firewalls have failed is today's complex topologies. In our metaphor, every exposed host—which includes traveling laptops and smartphones—is itself a gate, with a poorly specified policy and ordinarily complex code implementing it.)

Strong walls and good gatekeepers alone do not solve the brittleness problem; their function is to minimize the chances of a penetration. To take the next step, we need to isolate the complexity even more. That is, we need to limit the information flow to a complex (i.e., untrustworthy) section of the system. A good example is a typical web site, where the web server is a front-end for a database. Web servers are complicated, and not easily shielded by a strong wall; their function—accepting HTTP and running arcane scripts to generate HTML—is inherently complex. The database requests they make, however, are fairly simple. If we implement the gate properly, we can even deflect SQL injection attacks. Note carefully the role of the policy specification: if it says "allow a character string", it won't help; if, on the other hand, it specifies the allowable query types and the precise syntax of each argument to each query, we're in much better shape. Also note that syntax-checking can be very simple, if done via formal grammars and parser generators.

Design principles like these don't eliminate the need for walls and gates. However, they do teach us where to focus our attention.