ClearText

Security Think

Steven M. Bellovin
Columbia University

There are lots of problems with security education. The most serious, of course, is that we're teaching security to the wrong people; we spend our time educating security specialists, rather than the application and system builders who are quite inadvertently creating the security holes that plague us. But there are problems even with what we teach specialists: we spend too much time on things like cryptography and too little on how to think about security.

Don't get me wrong; cryptography is important. Students do need to learn about it, including the very practical fact that almost no one should ever invent his or her own crypto. Students also need to learn about firewalls, access control lists, passwords, and a plethora of other primitives. But that's the easy part.

The harder part is putting all of this together. A security specialist is rarely told to "figure out how to secure this TCP connection"; usually, the proper response is something like "use TLS" (or, in some cases, "enable the TLS option in the application"), at which point most of the solution can be left to any programmer who can read the documentation. Rather, the challenging questions are things like, "How do we secure this distributed application?", when it runs on several dozen different computers, requires a number of canned applications (each of which has its own security model that doesn't happen to match what you're doing), and has to talk to a legacy program written in COBOL during the last Ice Age.

Jobs like this, or the closely related task of evaluating a proposed architecture, are at the heart of what security people should do. It's a tough subject to teach, though: real systems are very complex, and each one is different. It could, presumably, be taught via case studies, but fully documented (and public) design details are rare. I have yet to see a textbook that even tries.

But even that pales before what most people find to be the really hard problem: thinking like a bad guy. It's easy to say "make users pick strong passwords," "watch out for buffer overflows," or "avoid cross-site scripting attacks," although these instructions seem to be difficult to carry out. How do you teach people to look for connections between two independent components that together can result in a vulnerability? How do you teach people to develop a sense that some component is more fragile and hence more likely to fail? How do you teach them how to balance usability and security, to ensure that users or your own people don't inadvertently sabotage security?

We see security failures in the physical world all the time. Recently, I wanted to buy a router bit from a large chain store. Naturally, because bits are small and expensive, a security person decreed that they be displayed in a locked rack. The floor staff I asked for assistance couldn't find someone with the key, so one of them just pulled out a knife and sliced through the plastic. I paid at a self-service register, declining a plastic bag (it was one small item) and thereby bypassing the gadget that deactivated the antitheft transponder. The alarm sounded when I walked out the door, but no one paid attention, perhaps because they were too accustomed to false alarms. Were there several independent security failures here? Of course! Did the entire system fail? Arguably, it did not—is the overall loss rate from theft of router bits more or less than what a fix would cost? That's the challenge: getting all of the trade-offs right in a complex configuration with many moving parts.

Teaching the basics is fine; anyone in the business needs to know them. But teaching people how to protect complex systems—"security think" —is the real issue

---