CryptoCorner
Editors: Peter Gutmann, pgut001@cs.auckland.ac.nz
David Naccache, david.naccache@gemplus.com
Charles C. Palmer, ccpalmer@us.ibm.com

Security Usability

In the security community, we've always recognized that our security proposals come with certain costs in terms of usability. Traditionally, that's the compromise we make to be secure.

Peter Gutmann
University of Auckland

Ian Grigg
Systemics

But the market has ruled against us. Time and time again, our fielded secure systems are ignored, bypassed, turned off, or constrained to such a small part of the process that the security result is practically nonexistent. Even worse for our mental self-satisfaction, those systems that claim to deliver security to users simply don't pass muster—they're not what we'd like to think of as secure systems.

How did this happen?

The security expert's frequent lament is the way in which security is bolted onto an application as an afterthought. It seems like an attempt to sprinkle magic fairy pixie dust over the product before it ships, and its lack of scientific quality makes it easy for experts to ignore and—even worse—denigrate.

In elevating the importance of secure practices and downgrading any considerations of usability, the security community has committed the exact same sin in reverse. If we consider security usability at all, we place it firmly in second place, and anyone wishing to dispute this claim is invited to try setting up an IPsec tunnel via a firewall or securing their email with S/MIME.

As a result, we spent the 1990s building and deploying security that wasn't really needed (at least, its presence didn't affect Joe Sixpack), and now that it's actually desirable (for viruses, worms, phishing, and so on), we're finding that nobody can use it. Thus, to properly deliver security, we must scrap the assumption of a usability compromise. The primary goal for current security efforts shouldn't be to further refine how many key bits can fit on the head of a pin, but to figure out how to make the existing stuff usable.

What relationship should we strive for?

Must usability and security work together, or should usability dominate security? We have a handful of choices:

- The two should work together as equal partners.
- Security comes first, and usability should be the compromising junior partner.
- Usability comes first, and security should be the compromising junior partner.
- Security is best left as a separate product, naturally layered into the application without disturbing it and without compromising strong design principles.

These choices change our architecture, the way in which we deploy secure systems, and the way in which security is delivered to (and experienced by) users so dramatically that we can't answer the trade-off question yet. However, experience with current mass-appeal software seems to indicate that security must follow usability, not the other way around.

Consider two contemporary examples, peer-to-peer (P2P) and the Internet phone application voice over IP (VoIP). Attempts to build intrinsically secure applications such as these have existed for some years, examples being Freenet (www.freenetproject.org), Nautilus (ftp://ftp.funet.fi/pub/crypt/utilities/phone/), and PGPfone (www.pgpi.org/products/pgpfone/).[1] (Nautilus goes back more than a decade, and there were other attempts at it before that). However, the really successful applications such as Gnutella (www.gnutella.com), Bit Torrent (www.bittorrent.com), and Skype (www.skype.com) started out—and gained widespread acceptance—as extremely easy-to-use (but generally insecure) applications that slowly bolted on security over time.

Post hoc security

The security mechanisms employed in emerging systems are often homebrewed as an afterthought, and the result is the widespread adoption of a somewhat-secure system as opposed to a very secure system that's not used at all. So if this is the best way to get a secure system deployed—should we design security

so that it can be conveniently bolted on after the fact?

Post hoc security operates as if the cycles of design, redesign, trial, and retrial are so uncertain and experimental that any attention to security during the process is likely to drain precious resources from the key issue: finding out what it is that gets a successful killer application accepted in the first place. Indeed, the marketplace reflects this with all kinds of flagship security products. SSL is a layered product for HTTP, for example, and PGP could be treated as a layered protocol for email. Relatively rarely is a tool conceived from the ground up to be both secure and groundbreaking; even SSH was a drop-in replacement for Telnet and the Berkeley r* utilities `rlogin`, `rsh`, and `rcp`.

To layer or not to layer

If we constrain ourselves to doing security after the fact, how should we do it?

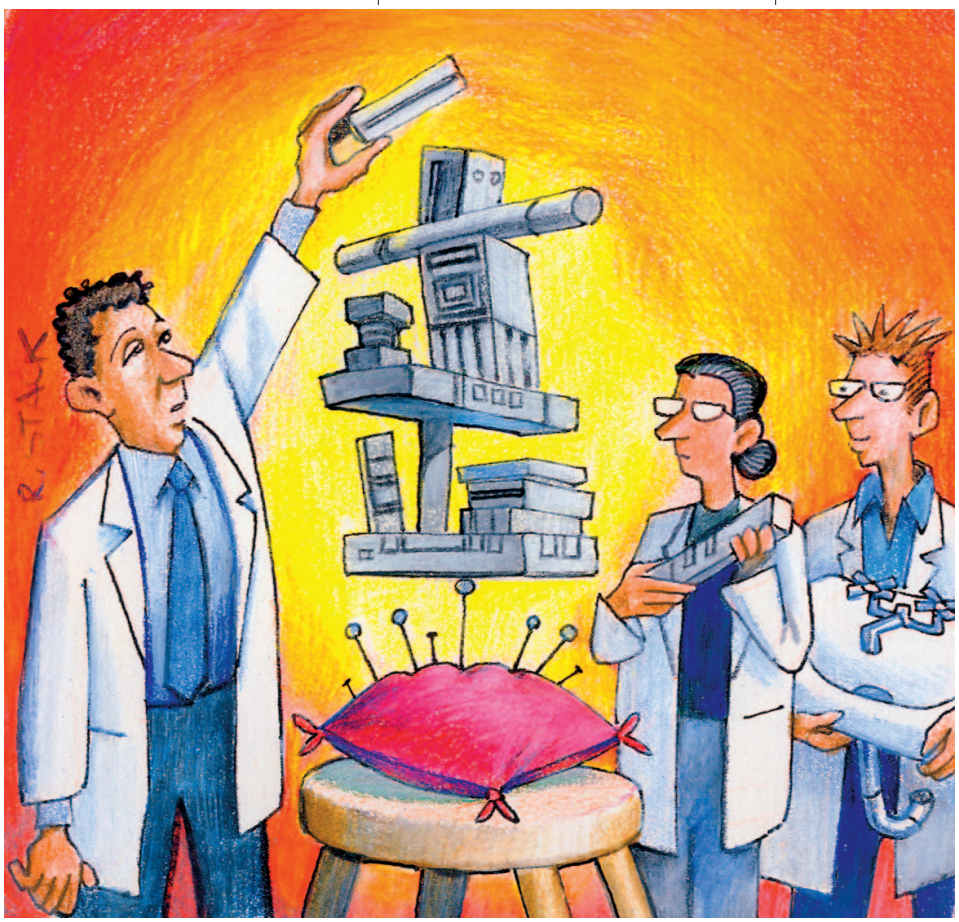
Let's look at the track records of layered security products as an option. The volume of mail secured through SMTP tunneled over SSL/TLS, for example, exceeded that of all other email security mechanisms combined—by an order of magnitude—within a year of its introduction because setting up and using other mechanisms (typically, S/MIME and PGP) is so painful.

Similarly, SSH, which was originally created as a secure Telnet/r* application, is now widely used as a universal secure-tunnel wrapper for insecure protocols because it's easier to bolt on the SSH tunnel than it is to use the secure form of the protocol being tunneled—if one even exists. SSH support is now almost mandatory for (Windows) FTP applications because wrapping the data transfer in SSH is the easiest way to secure it, even though the implementations of the SSH protocol in the application are usually minimal and often awful. As a result, users see a familiar Windows Explorer-style interface when they move files around, but under the surface, it's all secured with SSH.

Bolting on usability

Although it's common knowledge that you shouldn't bolt security on after the fact, it's even harder to bolt usability on after the fact. The bolt-on front ends intended to make hard-core crypto applications easier to use are probably some of the most complex applications the typical user will ever encounter. Moreover, they usually mask the arcane and convoluted mechanisms of public key cryptography key management, offering all this complexity and security jargon for the debatable benefit of securing that which is already in use daily, albeit with some minor statistical risk.

Let's look at the cost–benefit trade-offs of various commonly performed computer tasks. Average users can handle the red-eye elimination wizard in a photo editor, both because it's easy to use and because they don't want photos in which their kids look like vampires. They can balance their checkbooks with Quicken (again, through a combination of ease of use and incentive) and type up a basic letter in Wordpad or Word (the latter being a good example of infinite, but well-hidden, complexity). In all these cases, there's a perceived value in dealing with a slightly more complex interface: a little more complexity is acceptable for a fair offering in value. In contrast, a typical security-first application presents a small perceived advantage in exchange for dealing with an extraordinarily complex interface that would challenge the average computer science graduate.



Consider how the average user sends secure email. An example of a typical front end for doing this is the KGPG graphical front end to the Gnu Privacy Guard (GPG; www.gnupg.org), a widely-used implementation of the OpenPGP standard. Even though it's justly lauded for advances in usability, KGPG still requires users to perform laborious manual key management, often involving complex multitab dialogs filled with incomprehensible options.[2]

Matters are far worse in the standardized public-key infrastructure (PKI) world. To initialize KGPG, the user follows five steps in sequence, but a popular PKI certification authority would make that same user fill out 11 pages of incomprehensible information to generate a X.509 public/private key pair.[3] Send a secure email? No thanks, I can already send an email, and a secure one isn't worth that much to me.

We spent the 1990s building and deploying security that wasn't really needed, and now that it's actually desirable, we're finding that nobody can use it.

Security usability circa 1883

Auguste Kerckhoffs, a Dutch cryptographer who taught in France in the late 19th century, wrote an influential article that expounded on six basic principles of a communications security system:[4]

1. The system must be practically, if not mathematically, indecipherable.
2. It must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience (often referred to as *Kerckhoffs' law*).
3. Its key must be communicable and retainable without the help of written notes and changeable or modifiable at the will of the correspondents.
4. It must be compatible with the means of communication (most security mechanisms result in message expansion and transform text into nontextual data).
5. It must be portable, and its usage and function must not require the concourse of several people (consider what happens if you log onto a banking site from computer B when your keys are stored on computer A).
6. Given the circumstances that command its application, the system must be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe.

These observations aren't new, although they're apparently more easily forgotten than understood. It's interesting that of these six principles, four of them speak to usability whereas only two speak to what we would call secure practices.

Principles 1 and 2 are the ones that cryptographers are most enamored of, but based on our market experience, Principle 6 could well be the most important one. The system must be simple enough to be used (no doubt, one or more security systems or protocols that fail this principle immediately spring to mind). Indeed, if we were to take any pair of principles and rank them according to which ones we'd rather compromise, Kerckhoffs has them ordered in importance from Principle 6 down to Principle 1.

In a future article, we'll discuss in greater depth what has worked and what hasn't. □

References

1. M. Caloyannides, "Speech Privacy Technophobes Need Not Apply," *IEEE Security & Privacy*, vol. 2, no. 5, 2004, pp. 86–87.
2. T. Chance, "KDE Developers, Usability Experts Complement Each Other," NewsForge, 9 June 2005; http://programming.newsforge.com/programming/05/05/05/1823209.shtml?tid=25&tid=130.
3. P. Gutmann, "Plug-and-Play PKI: A PKI Your Mother Can Use," *Proc. 12th Usenix Security Symp.*, Usenix Assoc., 2003, pp. 45–58.
4. A. Kerckhoffs, "La Cryptographie Militaire," (in French), *J. des Sciences Militaires*, vol. IX, Jan. 1883, pp. 5–38.

Peter Gutmann *is a researcher at the Department of Computer Science, University of Auckland, New Zealand. His research interests include security engineering and the practical application of cryptography. Gutmann has a PhD in computer science from the University of Auckland. He is a member of the IEEE, ACM, and IACR. Contact him at pgut001@cs.auckland.ac.nz.*

Ian Grigg *is a financial cryptographer with Systemics, a supplier of payments and trading systems. His interests include economics and governance of Internet financial systems, and applying cryptography to end-user problems. Grigg studied computer science at the University of New South Wales and has an MBA from London Business School. Contact him at iang@systemics.com.*

Interested in writing for this department? Please contact editors Peter Gutmann (pgut001@cs.auckland.ac.nz), David Naccache (david.naccache@gemplus.com), or Charles C. Palmer (ccpalmer@us.ibm.com).