Toward Usable Security

Dirk Balfanz, Palo Alto Research Center

Traditionally, security and usability have been considered to be a design trade-off, where the user ends up with one or the other, but not both. We believe that it is not just possible to combine these two properties in one system, it is in fact necessary to provide more usable systems in order to make them truly secure. The following three problems need to be tackled before we can have systems that are both usable and secure.

1. Trust Models. Users understand that if they lock something in a physical safe, it's reasonably hard to get at. Users typically do not understand what it means for Verisign to sign a certificate that contains such items as a public key (which is a large number), Basic Constraints, expiration date, and a Distinguished Name. Yet they are asked to make trust decisions based on digital certificates. This disconnect between underlying technology and trust models users understand is one of the reasons why public key infrastructures still do not exist on a large scale. Other trust models, such as PGP, are even more complicated, and are therefore used by even less people. Here are some ideas how this challenge could be tackled:

- Rely on models users understand. One could imagine, for example, a computer that would allow access to other computers based on proximity. If the other computer is in the same home, then access is granted, otherwise it is denied. This allows users to build a home-area network while understanding that it is necessary and sufficient to break into their home in order to gain access to the network. This model puts virtual walls where physical ones exist.
- Stop relying on global infrastructures of public-key identity certificates. Issuing identity certificates just shifts the question from "should I trust this key" to "should I trust this name". Again, users are more likely to trust everything in their home, everything they own, *etc.* Instead of crafting systems around a global PKI, craft them around trust models users understand. This model reflects users' beliefs their own devices are trustworthy.

2. Human-Computer Interaction. Once we understand what the trust models are that users are comfortable with, we need to find models for user interaction with the computers and their devices (this is analogous to defining that "mouse click", "opening an item on the desktop", *etc.* are the fundamental building blocks of the HCI model used to interact with common office applications). This certainly includes judicious GUI design, but is not limited to that. We believe that much of the traditional security GUI can safely be abandoned in favor of what we call "implicit security". Much of the security configuration of a computing system can be inferred from user actions that are not necessarily expressed in terms of security. For example, when a user sends an email to Alice@foo.com, we can assume that the intended recipient is Alice, and only Alice, and cryptographically ensure this without the need for additional GUI.

We found that simple gestures (*e.g.*, pointing a PDA at a computer) can be used to "piggy-back" security (*e.g.*, exchange public keys between the PDA and the computer, which can later be used to establish a secure connection between the PDA and the computer). We need to find more of these HCI primitives that allow users to intuitively express their intent, and at the same time allow security to be inferred.

3. Meeting Users' Expectations. It is often hard to describe what exactly is considered a "secure" system. For example, is the fact that Outlook warns its users before they open an executable attachment a sign that Outlook is secure, or should it be considered insecure because it lets users catch dangerous email viruses. In general, it is up to each individual user to decide what constitutes a breach of security. What system designers need to do is make sure that the consequences of their actions are apparent to users, and that users can effectively express their intent and see the resulting security consequences. Revisiting an example from above, when sending an email message to Alice@foo.com, the consequence most likely anticipated by the user is that Alice, and no one else, will be receiving this message. Therefore, the system should automatically encrypt the message to make sure that indeed only Alice can read it. This fundamentally affects the system design: For example, the system somehow (without bothering the user) needs to find out Alice's public key, and Alice needs to have a corresponding private key, necessitating the participation in some sort of public key infrastructure. New technology like Identity-Based Cryptography can help here. While it is easy to see how unanticipated consequences can be avoided in certain situations like the one above, it's a great challenge to understand what users expect to happen, and then to design a full general-purpose computing system that avoids unanticipated consequences.

Further Reading: You can find a bibliography of (among other things) papers describing some of our work in this area at `http://www.parc.com/balfanz`.