

The User Experience of Computer Security

Luke Church, luke@church.name, SHB '09

[DRAFT: 2009-04-07]

Introduction

Ross's blog postings (Anderson, 2008) from SHB '08 summarised my presentation with the phrase *"In fact, this is a field in which we really need some ideas!"* A year later, this paper continues in the same theme, each section provides more questions than answers, but with some glimmers of progress.

Most of this paper will look at *experiential* aspects of security. As we move towards systems which allow genuine user control over information, it's not acceptable to build systems that are unusable, but neither is it sufficient to build systems where users achieve can achieve the technical configuration they desire, but don't have confidence they have done.

This might seem like an odd way round to do things – why try to solve the experiential problems, before we've solved the usability problems? As we shall see, I don't believe it's that simple. In problems like privacy, the user experience and usability cannot be determined independently.

After discussing this, and a number of factors that seem to affect the user experience, I'll comment briefly on a year long experience of doing security usability design in industry and the need for better ways of talking about usability before user studies.

The User Experience of Privacy and Security

What are current user experiences of privacy and security like? In many cases, fairly problematic. Let's take privacy: At one extreme the experience is of essentially no control. The user is presented with a privacy policy which in theory they can accept or decline. (Hoofnagle & King, 2008) show that users consider the *presence* of a policy to

"create the right to require a website to delete personal information upon request, a general right to sue for damages, a right to be informed of security breaches, a right to assistance if identity theft occurs, and a right to access and correct data the mere fact that a website has a privacy policy, and assume that websites carrying the label have strong, default rules to protect personal data"

Even if this wasn't the case, this is hardly a negotiative experience.

Other organisations try a different approach. For example, Facebook offer a moderately fine-grained access control system with 61 drop down boxes across seven separate pages¹. These options migrate over time, so the user has to keep their settings up to date against a moving target (Anderson et al Stajano, 2009).

Our preliminary analysis of Facebook's Privacy UI (In progress: Church et al., 2009), using a combination of analytical usability tools such as the Cognitive Dimensions of Notations (Green &

¹ As of 14th of March, 2009

Petre, 1996) , and empirical studies incline towards the unsurprising conclusion that the usability of such systems is poor, leaving users with behaviour they didn't desire, and no experience of control.

One way of interpreting this kind of position is that Facebook doesn't care about their users' privacy, or even that they would prefer their users not to be able to express their privacy wishes. While this may be true, we should at least suspend judgement until we have produced a real, viable alternative. Afterall, Facebook are not alone in facing this challenge, other user interfaces have faced similar challenges. If we, as a community, are going to have a genuine privacy debate we must be more constructive than this.

Our discussion (In progress: Church et al., 2009) is one starting proposal for a constructive debate. We acknowledge that the Facebook UI is a form of programming language. We then apply the design methodologies of end user programming to design an alternative. The reconstructed interface creates a 'privacy story'. A pseudo-English description of the users' privacy policy, it is written in a controlled vocabulary, using a structured editor. It has a visualisation function that allows the user to see the effects of their policy on who can see what.

[INSERT Final Design Sketches here]

Analytically, our UI will do better for people engaging in *Exploratory Design*, (changing the information structures, exploring their way to an answer). We have achieved a user interface with lower *diffuseness*, lower *viscosity*, a better *abstraction gradient*², better *closeness of mapping*³ etc. Analytically then, we have a 'better' user interface – but one that is now unmistakably programming.

We hypothesise that acknowledging the programming nature of the problem and designing for it explicitly, far from the standard concern that users will somehow be cognitively unable to cope, will result in users being able to express their needs more accurately and feeling more in control whilst doing so. Experimental work to test this hypothesis is starting, and we expect initial results soon.

There is another interesting design possibility that acknowledging that privacy configuration is an act of programming allows – sharing. Just as users can share programs/scripts that they create on their home computers, so they can share their privacy policies, in a suitably sanitised form. Others can adopt their policies, customising them as needed. Supporting social mechanisms in this way provides users with the ability to choose how much effort they would like to invest in managing their privacy. If they are content with 'expert settings' provided by either Facebook, or a trusted user, then they can do so. However, if they wish to take more control they can customise or write their own policies.

This, finally, begins the construction of a nuanced response Whitten's Secondary Goal Property (Whitten, 2004), by allowing a spectrum of investments of attention

The User Experience of Perfection

The Facebook example has an interesting assumption built into it – people do better building things in an exploratory manner than they do by thinking really hard about the problem before starting. This bears a little more attention.

² The rate at which the user has to learn concepts

³ The correspondence between the problem domain and the abstractions that are presented

I teach Software Design to Cambridge Computer Science undergraduates. We talk about the difficulties of requirements analysis and the failure of the waterfall method⁴ – customers find it essentially impossible to specify what they would like without having a concrete artefact to design around. We then discuss iterative methods which allow ‘rough cuts’ and refinement. They, generally, successfully complete a design exercise using iterative methods and usually seem fairly persuaded by the benefits.

I then set them a task to design a life critical system, say an aircraft control system and cockpit, stressing the importance of system. They, almost without exception, revert to trying to understand the whole of the problem before beginning the design task. When challenged, they acknowledge that this design methodology is unlikely to result in perfect interfaces, but somehow feels ‘safer’ than the more iterative approach.

This anecdotal result resonates through the technical and social practices of Computer Science. At a technical level, type systems induce *premature commitment*, forcing us to think about the long term design of systems rather than building them iteratively. At a social level, one of largest challenges we faced in designing our proposed changes to the Common Criteria (Church et al., 2008) was aligning its desire for heavy up front design with iterative design methodologies.

This is not simply a mistake of Computer Science; cognitively what is happening is known as *Useful Awkwardness*. Forcing people to think hard about their problem, and recasting it into another form, results in certain types of mistakes becoming apparent. It is a common belief that forcing people to think hard about their security decisions before implementing them is a good idea, for this reason. But too far along this direction, and we’ll be encouraging mistakes and running into the Secondary Goal property problems. Assuming that ‘end users’ should work in same way computer scientists seems somewhat problematic.

This raises questions: *How usable should we make our security/privacy systems? In what ways should we seek to introduce Useful Awkwardness? How do we do so without creating an unpleasant User Experience?*

User Experiences of Simplicity

Simplicity holds an interesting place in many of the discussions of technology. It is often desired by users, often claimed by technologists⁵ and promoted by evangelists (Maeda, 2006). However, it is not clear that they mean the same thing - this has affects on the way in which the abstractions of a piece of technology affect its usability.

To make this more concrete, let’s consider a few abstractions: first, encryption/decryption. This concept seems to work fairly well, in very anecdotal tests descriptions that technically naive users gave were reasonable accurate, along the lines of “*information in a form so that cannot be accessed without a password*” – we might generalise ‘password’ to ‘secret’, but otherwise this is more of less reasonable.

⁴ The waterfall method is a linear model for software development, where you determine the client’s requirements in advance, refine them into a design, implement and release to the customer

⁵ E.g. Apple: “No other operating system – Vista included – offer the innovation and simplicity of Mac OS X”: www.apple.com/getamac/whymac/

The point to note is that it is pretty much a concept solely within the mathematical domain; there is nothing social about the nature of the encryption. I would assert that the abstractions of modern security are fairly good in this regard. To a first approximation our encryption algorithms currently seem uncrackable.

At the alternative end there are notions like 'security', which is used as a single word/concept, but is somewhat difficult to define. From an end users' perspective, a secure system might almost be defined as "a system that does what I want it to".

These are two extremes, it's worth looking at something in the middle to understand a little better what's happening. Let's consider a secret sharing scheme, and how it might be used.

Secret Sharing is a cryptographic system wherein a secret, a large number, is broken up into a number of pieces using a mathematical function so that constraints can be placed on how many of such pieces need bringing together in order to reassemble the secret. This is usually expressed in the form 'k of n' – e.g. 3 of 5, implies that the secret was broken up into 5 pieces, and any 3 of them can be used to reconstruct it.

The underlying mathematics of how to achieve this might be slightly devious, but it at least appears to be conceptually simple. However, when one looks at how they might be used in practice, the simplicity evaporates somewhat.

One common way in which to use these schemes is to decide the K/N, and put the secrets on smart cards protected by passwords. These are then given to different people, and used to enforce the number of people required to authorise a decision. The use of cryptography gives some insulation against software mistakes, and therefore some sense of higher security.

However, a number of practical problems arise – for example, we've had to *prematurely commit* to the K/N. In a naive implementation, if we need to add another person to the group, then we have to bring the cards back, decrypt the secret and re-encrypt it for placement on new cards. This is often impractical. In other cases it's often necessary to be able to keep a record of *who* authorised an operation, not just that enough people did.

The end result of the increasing numbers of these requirements is that the system turns into 'cards are used for *authentication*, not *authorisation*' and software is used to bridge the gap and determine whether an operation should take place. Somewhat shifting the security behaviours that were originally intended.

Cryptography seems to be like this, high level primitives are too brittle to cope with the social complexities. This creates an odd instinct:

If you have what appears to be a simple system for enforcing a property in the social domain, rather than the mathematical domain, one should be suspicious of whether it will be flexible enough.

However, if we abandon simplicity as a technical heuristic, is there anything that we can replace it with? How should we measure closeness of mapping of social abstractions?

Accelerating Usability in Industry: Analytical and Inspection Techniques

Some time reading the proceedings of security usability conferences, such as SOUPS, shows a trend towards empirical studies. A system is designed, and then deployed in either a lab and/or a field trial in which its usability and security behaviours are monitored. The results are then reported.

Clearly, empirical studies of usability bring benefits and certainly have their place – however they're not the only usability evaluation technique. In human computer interaction, they're often coupled with 'inspection methods' or 'analytical techniques', these are intended to predict usability issues and to provide a framework in which to discuss usability properties. We have already seen some elements of such a framework with the discussion of properties like 'premature commitment', 'hidden dependencies' and 'useful awkwardness' etc.

My experience of the past year doing security usability work in industry has highlighted the need for such analytical techniques, to complement user studies.

Firstly, there is a need for a framework in which to consider usability implications long before the product exists. The standard user centred design techniques for this revolve around prototyping, which – even in its lowest fidelity forms – is still time consuming and often doesn't address questions of the usability implications of abstract models. E.g. Should I choose to group my ACLs this way or that? You can't do a user test to empirically decide each such question – but it would be nice to be able to make decisions on something other than faith in a design instinct.

Secondly, user studies are very expensive, and often have commercial sensitivities. Companies engaged in security work often need to present a 'professional' image which dissuades them from doing low fidelity prototyping. Marketing and sales people are reluctant to let customers see potential for new products, in case they delay purchasing of existing ones, and to ensure that the 'corporate message' is maintained.

Thirdly, product lifecycles are not indefinite. Usability, like security, often needs to feed into the fundamental model of the technology. It's not usually commercially realistic to stall the development of the core technology for six months whilst prototyping studies are carried out. Mandating empirical studies, somewhat relegates usability to designing GUIs – by which time it's often too late to 'retrofit' usability to the conceptual models.

Forthly, security usability problems are often not short term ones. Usability issues often arise once the user has left the 'learning phase' – these are hard to capture in empirical studies. For example, what mistakes will users make after they've settled into the system for six months? What problems will they have changing their security policies to organisational changes, as opposed to designing a new one?

Fifthly, usability studies have an over-fitting problem. A user study of one, ideally tells you the usability properties of the system for that person. Are they general problems, or specific to that person? How can we generalise the results? One approach is to do larger studies – but that exacerbates the above problems.

So – we need ways of talking analytically about security usability in order to address the above. There are a few candidates, Whitten's Making Security Usable could be considered to be a start, as

could Green's Cognitive Dimensions – but these highlight an interesting problem in themselves. Whilst they have achieved academic success, they have had relatively restricted uptake by industry – in the case of the Cognitive Dimensions this seems to be due to difficulties in applying the framework by practitioners. (Green and Whitten, personal communications)

This leaves us with an interesting design challenge – we need analytical frameworks for discussing security usability that are concrete enough to be easy to apply, avoid the trap of abstracting away the key properties of the users (Blackwell et al., 2008) – but abstract enough to be rapidly applicable very early in the design phase, are predictive of real problems, and are suitable for having balanced discussions about tradeoffs.

This still seems like a field in need of some ideas.

Bibliography

Anderson, J., Bonneau, J., Church, L., & Stajano, F. (2009). Privacy Suites: Shared Privacy for Social Networks. *Submitted to Workshop on Social Networks, SIGCOMM*.

Anderson, R. (2008, June 30). *SHB Blog*. Retrieved April 7, 2009, from Light Blue Touchpaper: <http://www.lightbluetouchpaper.org/2008/06/30/security-psychology/>

Blackwell, A., Church, L., & Green, T. (2008). The abstract is 'an enemy': Alternative perspectives to computational thinking. *Proceedings of PPIG'08; 20th annual workshop of the Psychology of Programming Interest Group*, (pp. 34-43). Lancaster.

Church, L., Anderson, J., & Bonneau, J. (2009). (In progress) Shared, Intentional Privacy Policies. *SOUPS Poster Track*.

Church, L., Kreeger, M., & Streets, M. (2008). Introducing Usability to the Common Criteria. *International Common Criteria Conference*.

Green, T. R. (1989). Cognitive dimensions of notations. In A. Sutcliffe, & M. L. *People and Computers V* (pp. 443-460). Cambridge: Cambridge University Press.

Green, T., & Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 131-174.

Hoofnagle, C. J., & King, J. (2008, September 3). *What Californians Understand about Privacy Online*. Retrieved from Social Sciences Research Network: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1262130

Maeda, J. (2006). *The Laws of Simplicity*. Cambridge, MA: MIT Press.

Whitten, A. (2004). *Making Security Usable (PhD Thesis, CMU-CS-04-135)*.