

# End User Security: The democratisation of security usability

Luke Church  
Computer Laboratory  
Cambridge University, UK  
luke@church.name

## Abstract

To date, the usability of security systems has been problematic. I argue that this is partially a cultural problem within security research, partially a result of a conflation of the abstracted computational world with the complex social one, and partially a result of security researchers focussing on mechanism usability rather than the usability of systems within their social context. We present End User Development as a possible route forwards and show how its lessons can explain and generalise parts of the existing research in HCI-SEC (Human Computer Interaction – Security). Finally, we consider some challenges that remain in applying End User Development to security.

## 1. Introduction: The state we're in

It's now a truism in security engineering that 'usability is a problem'. The volume of work in HCI-SEC is growing rapidly, but many problems remain: people fail to understand their privacy settings on social networks, corporate data still gets lost because of bad Access Control Lists<sup>1</sup> and passwords still get written on post-it notes. Generally, the security state of end user PCs is poor. [REF: Microsoft security ecology report]

Further, psychology based attacks have migrated from the periphery to centre stage. The all but forgotten Anna Kournikova and Love Bug viruses stood out because of their human aspects, as opposed to their technical ones. Phishing is now a major concern of security engineers, the security of social networks is a growing problem, privacy issues remains unsolved. In order to consider how we might start addressing this asymmetry in the human aspects of security systems, we shall first consider the anatomy of a simple system.

To take an example from the medical domain: security systems are used to attribute actions to particular medical practitioners. Such a system might be implemented by having the practitioner insert a smart card into a computer before authorising an action, and having the computer log the whole operation.

We'll view this scenario in the following, rather simplified, manner:

The situation consists of a model, a policy and a mechanism. The *model* is the computational representation of the world, in this case, it might include a

---

<sup>1</sup> Bishop [REF] asserted that more than 90% of security problems are caused by configuration errors. Whilst the figure may have changed, it seems likely that many such problems remain, given that the complexity of our systems has increased and the average user competence has, at best, remained constant

representation of a ‘user’ (typically representing the user’s smart card<sup>2</sup>), the notion of an ‘activity’, such as setting the morphine dose to a particular value, and the notion of an authorisation event, which occurs the user performs some task.

The *policy* is a set of rules that the system is to enforce. An example might be ‘no activity may be performed without an authorising event, performed by an authenticated user’.

The *mechanisms* are the apparatus that is used to ensure that the policy is enforced. In this case a computer with a card reader and a user interface that allows the practitioner to authorise actions.

As suggested above, in recent years there has been concern about the usability of security systems such as these, particularly focussed around the mechanisms. In this paper we shall first consider mechanism usability, the mainstay of HCI-SEC. We shall argue that focussing on mechanisms is necessary but not sufficient, and go on to consider social aspects. This will lead us to a reconceptualisation of security configuration as End User Development and we shall consider the tradeoffs of doing so.

## 2. Mechanism Usability

Mechanism usability studies have been a core focus of HCI-SEC. Such studies are generally focussed around experiments. A typical experiment will concern some micro-task, say ‘authenticate yourself to the system’. Proposed systems for achieving this task will then be compared with a baseline representing current practice, such as ‘password entry’. Quantitative measures include as ‘the number of times password had to be reset’ or ‘time taken to log in’, null-hypothesis tests are then performed to see whether a statistically significant difference between the baseline and the proposed mechanism can be found.

Such studies can be found throughout the literature for assessment of password generation mechanisms [REF], graphical password usage [REF], PKI user admissions [REF] etc. In some cases similar techniques have been used to measure the usability of applications, where the tasks become things like ‘send an encrypted email to Joe’. The famous study into PGP, [REF: Why Johnny can’t encrypt] was carried out along similar lines, using successful task completion as its numerical measure. [TODO: Check this]

When such an experiment is being designed, there are two types of validity to consider, internal and external. Internal validity is the concern of the scientist about the quality of the experiment, for example how repeatable it is. External validity is the concern of the user about whether the study generalises to explain other circumstances, for example, how well it explains a real phenomenon.

The techniques of assessing mechanism usability were derived from experimental psychology and as such stress internal validity. Security research is concerned with the correctness of systems such as cryptographic protocols as compared to theoretical ideals. Hence it is perhaps unsurprising that these methods for assessing usability, have been persuasive to security researchers.

---

<sup>2</sup> If the reader is disturbed by this conflation of the human with their authentication token, I have complete sympathy. However, in my experience, this is common in real systems. As we shall see later, this is not just a rhetorical point, but a significant cause of problems.

The most recent trend in this theme of modelling an individual's behaviour is the consideration of behavioural economics. This branch of economics discusses cognitive 'heuristics and biases'[REF], factors that affect human decision making. For example, suggesting a possible mechanism for how something might have occurred results in subjects perceiving the outcome as being disproportionately more probable. This is important for security, as it may be possible for attackers and defenders to change the action that a user will take by using these heuristics. For example, in phishing sites the text 'your security image is not available whilst we upgrade security' sometimes appears. This provides users with a plausible reason why the image isn't there and may make them less likely to believe they are being phished. [REF?]

I have no doubt that mechanism and micro-behavioural studies are important for security usability; the early studies [REF][REF] effectively highlighted the existence of usability problems, newer ones are helping iterative improvement of the designs of CAPTCHAs and phishing resistance mechanisms. However, such mechanism usability studies have a number of limitations:

- Mechanism studies tend to focus on internal validity, at the expense of external validity. For example password memorability is likely to be different if a new password scheme is used only in a lab setting as opposed to if the scheme is widely used, due to interference effects. Therefore lab experiments on passwords are often sound for the lab context, but not generalisable.
- The studies are strongly susceptible to the 'lure of the measurable' [REF]. I.e. researchers measure what is measurable rather than what is important.
- Mechanism studies typically don't capture usability issues that arise out of the social use of technology.

The last point warrants some discussion. Let us return to our example of the medical system. When we observe the work practices that actually occur on the ward we notice a complexity: the person who performs an action isn't always the person who decides what action is appropriate. For example a consultant may decide to increase the drug dosage for a patient and may instruct the registrar to do this. The nurse may then, whilst disagreeing with the action, log in to the computer and authorise the action. The system will record the registrar's authentication and authorisation of an action that they didn't agree with.

No amount of mechanism usability studies on the process of authenticating or authorising an action will be able to predict the existence of this problem as it is a difficulty that arises out of the context of use, not out of the mechanism. In order to make progress, we also need usability analysis techniques that can account for the social uses of technology.

### **3. Social Contexts**

All technology exists and is used in a social context. If this were not the case there would be little need for computer security. As suggested above, the social context in which software is used can have a substantial impact on its usability. In the example of medical authentication, I argue that it is the *model* that is at fault not the mechanism; the authentication mechanism serves to accurately identify the physical person who is performing the action, the logging mechanism faithfully records this.

However, the model contains an implicit assumption: the person *authorising* an action, and the person *performing* the action are the same. This isn't always true in the social context in which the technology is used.

There are four possible routes forward:

1. Ignore the issue: if anything goes wrong, blame the person who *performed* the action for authorising it
2. Change the people: 'redesign' the social processes such that they agree with the computational model
3. Accept the issue: when reading the audit logs, interpret the authenticated person as the person who *performed* the action, rather than the person who *authorised* the action
4. Change the technology: redesign the model (and probably the mechanisms), to take account of this behaviour

If the problem is not appreciated by the system designers options 1 or 2 may well occur; by contrast both options 3 and 4 require an awareness of the problem.

I assert that option 1, whilst frequently adopted in the security industry [REF][REF][REF], is unacceptable. The system is failing to meet its aim of providing appropriate accountability for actions. As we shall see in section [DOC REF], option 2 is problematic and often results in systems with poor usability. Option 3, while more acceptable than option 1, is still problematic; the system is again failing to provide appropriate accountability and requires that the readers of the logs be aware of the social context in which the technology was being used. Using the logs in this way may prove difficult in legal proceedings [REF: STS Study on court cases] further limiting the value of such logs.

This leaves us with redesigning the model and the related problem of how the designers and users (of both the system and the logs) gain the required awareness of the problem in the first place. The need to build systems to support use in context has been a strong force in the migration of HCI away from the individual and towards the social. Ethnographic techniques, such as field studies, are used to inform the design of systems; if we can understand how people actually perform their work, then we can build systems that support that reality. For example, we might be able to predict the existence of the above attribution issue by observing that in the ward medical practitioners often request that other practitioners perform actions on their behalf.

However, such ethnographic techniques are not without their problems:

- Different workplaces may have different social processes. This causes problems with generalising the knowledge gained and with the 'productization' of systems, i.e. selling the same system to multiple organisations
- Designing customisations for systems to better support the social processes might typically be done by software consultants. This provides a problematic incentives landscape, we shall discuss this further in section [DOC REF]
- Studies of social behaviour in organisations are hard to perform. Bringing back information about a work place that is both accurate and helpful for informing design requires that the investigator be sensitised to such issues. Such investigators are in short supply.

- It is unlikely, even with the support of ethnographers, that such a system will be ‘right first time’. Consequently iteration is likely to be needed. This will be discussed further in section [DOC REF]

Given these problems, I suggest that a change in perspective is needed. Instead of having one organisation build software to be deployed in another organisation, we should build frameworks and let the users themselves program their system. They are the most likely to know what *actually* happens within a social context. This perspective, End User Development of security, would:

- Mean that products are customised to their environment by a local domain expert
- Provide better incentive alignment; users of the systems are now the ones designing them
- Allow faster customisation and feedback loops

We will show later [DOC REF] that when we apply the lessons learnt from the usability of programming systems, we can explain and generalise much of the current security usability literature. Let’s now consider how we might build systems that support End User Development of security.

## 4. End User Security

This more speculative section looks at the possible ways in which End User Development (EUD) [REF] might be used for development. EUD is an approach which seeks to reduce or remove the hard distinction between developers and users. The idea is to allow part of the solution to be developed by the professional developers and part by the end users. Let us first consider what activities a user might engage in that could be considered to be programming, how we might apply EUD to security, and then consider the tradeoffs of doing so.

We have already observed in section [DOC REF] that the usability of security configuration, such as access control, is problematic. If we consider briefly what the process involves:

- The user is specifying how the system such manage abstract entities, such as files
- The behaviour being specified is how the system should handle future requests (e.g. permit/deny) from other entities, such as other users
- The permissions that apply are usually determined by a mechanism that includes inheritance and overriding

Under the definition in [REF, CHECK: First steps in programming], this could certainly be described as a programming activity. Similarly from the medical example that we have been using through this paper, the definition of the model and policy, which we have argued was crucial for usability, involve the description of abstract notions, (e.g. ‘an action’), and the relationships between them (e.g. ‘a policy governing an action’).

So, if we accept that some of these activities are programming-like, what might be the advantages and disadvantages of trying to build explicit support for the end user doing development?

## 5. Advantages and disadvantages of EUD approach

It seems likely that the process of switching to building systems that support EUD is going to be an expensive and difficult one, both technically and socially. Consequently we need fairly strong reasons for doing so. Let's first return to a discussion of what goes wrong with other perspectives on development by understanding how social and technical elements of computing interact.

### 5.1. Social aspects of use

Formal systems<sup>3</sup>, such as computers, can determine social ones. There are a growing range of examples, including:

- When the International Classification of Disease replaced geographic names (such as [XXX]) for diseases with systematic ones (such as [XXX]), patients records lost their implicit details on the patients travels
- [TODO: Check from Anderson] The US passport system mandates surnames, meaning that the cultures that don't have them have to be allocated new ones
- [TODO: Further examples]

Whilst these examples may seem irritating, more serious usability problems occur when the social environments become complex or the stakes are higher:

- The London Ambulance information system contained an assumption that crew took the vehicles that were allocated to them. The failure of the staff to do so (more senior staff took the newer vehicles) contributed to the failure of the system. [REF]
- In a timeshare oilrig timesheets and project codes were used to track how much time was spent on various processes. However various social workaround had been constructed to support projects that the accountants hadn't yet developed codes for. The introduction of a computer system that enforced correct codes caused disruption to these practices

However, this computation-social interaction has an another complexity: just as computational systems can influence social ones, so social systems can also influence the way computational ones are used. Imagine being in an office which institutes a 'you must lock your desktop when you leave your desk' policy. If you're the only person in the office who follows this new policy, then it sends unwelcome social signals about how much trust your fellow officemates. In this case social considerations are likely to override the 'appropriate security behaviour'.

---

<sup>3</sup> By formal, I mean that the fundamental mechanisms of are precisely defined. Humans also encounter formal systems in beauracies, sport scores, some diagrams etc. However most human conversation, including this article, is informal in the sense of not precisely defined.

Another example where technical systems mandate one usage ‘to be secure’, but social/psychological practices have a more significant force can be found in email handling. Even in the days of ‘Love Bug’ it was commonly appreciated that one shouldn’t open attachments, however emails with the subject line ‘I love you’ coming from people you knew were still likely to be opened [REF]. A recent manifestation of this is phishing emails appearing to come from people you know [REF].

So what happens when technology is used to determine social outcomes is complex. In some cases, like the lock your desktop policy, the requirement may be ignored. This effectively turns option 2 from section [DOC REF] into option 1. In other cases the new formal system may seriously disrupt work with the existing social processes, as in the London Ambulance case. Consequently it is important to understand the behaviours that occur as new technology is finding its place within an organisation.

### **5.1.1. Living with security: Adaptation and Negotiation**

Section [DOC REF Anthro] presented a picture of the problems of building systems without a good understanding of the social environment in which they were to be used.

I argue that one of the main benefits of supporting EUD is that, as a process, it is less likely to result in computational determinism. We’ll look more at what causes such determinism in the next section [DOC REF]; the primary advantage that an end user has is access to large amount of domain specific knowledge shape the system. This domain specific knowledge may consist of both the vocabulary used by users, as well as the tacit practices of that community. They can then build security mechanisms that support, rather than hinder, those practices. For example this developer is likely to have seen situations where a practitioner asked another to perform a task for them. A designer that is living with the user community is also more likely to think about practitioners and of the real work social situations rather than the computation abstracted versions. [TODO: Evidence for this from EUSES?]

However, whilst I stand by this perspective, much software isn’t built this way and yet the world doesn’t end. This is because social processes co-evolve with the technology. People form new social behaviours to accommodate the technology, whilst the sponsors of the technology work to support its deployment. Over time the working practices settle down to reach a new accommodation with the technology, be it a happy one or not.

However, it is not obvious how these social processes might work with a security system. Generally such systems, as mechanisms for enforcing a computational model, are designed to try and minimise such negotiation and mandate that their model is followed. The existence of such negotiation is sometimes even considered as a failure of the security system, as is the case in online gaming.[REF] Therefore, as we shall see in [DOC REF], there are both social and technical challenges in supporting negotiable security technology.

Let us now consider what happens when developers do think about their users in terms of abstract entities, rather than concrete examples of the users’ needs.

## **5.2 Social aspects of design**

### **5.2.1. Computational Thinking and security culture**

I argue that researchers and designers are most likely to cause computational determinism when we think in terms of constructs in the computer, as opposed to constructs of society and the ‘real’ world. If we think in these social terms, we are likely to be persistently reminded of the complexities of the world, however computational models seem seductively simple, as if they are extracting the essence of the problem and the rest can be dealt with later. Yet this is exactly the point, the complexity of social situations *is* the essence. That people authorise actions that they don’t agree with is not just an inconvenience, it is a serious issue for the usability of security systems. As we argue in [REF: The Abstract is the Enemy], abstracting away from this inconvenient complexity makes the security engineer’s life easier, but blinds them to the side effects of their simplified computational models.

But this presents a problem: to be able to build a computational model of a system which is required in security applications, we have to abstract. The key is not to do so blindly, as we shall see in section [DOC REF], the integration of new technologies is a social process. However unless we are aware of how our technology shapes the world, we are unlikely to understand how to best support this process. I argue later in section [DOC REF], that moving this process of abstraction closer to the final user is one of the benefits of end user programmer.

By contrast, I argue, the current culture of security research is likely to result in unwitting computational determinism. This is because such people are selected for their ability to think about the constructs of the computer. Much of ‘Security Science’<sup>4</sup> is about a competition between attacker and defender about who knows more about how a system works, with the attacker attempting to discover new ‘feature interactions’, and the defender attempting to predict and close such problems. This is an intensely technology centric activity, it seems unsurprising that this type of work will attract people who focus on computational abstractions, as opposed to their social counterparts. In the author’s experience, in casual conversation many security researchers take this to the extreme perspective of reconceptualising the world into abstracted computational metaphoric representations. There are even occasional attempts to do this in the literature, for example, [REF Secure Ceremonies].

Due to the social constraints as to what makes a good security researcher, security as a discipline is in more than usual danger of unwittingly causing computationally determinism.

An alternative approach to having the security researcher define the system is to build a configurable platform and have consultants customise it to meet the users’ needs. However, I argue in the next section [DOC REF], that consultants are also a problematic choice.

### **5.2.2. Incentives**

A key result from the security economics over the past few years has been that many systems fail because of incentives misalignment rather than technical reasons. I argue that EUD provides better incentive alignment than current development models.

---

<sup>4</sup> As opposed to security engineering, I use security science to discuss the more theoretical, research orientated, elements of security

Particularly the use of consultants is difficult because:

- They tend to be expensive. I argue that it should often take less time to teach a domain expert an EUD language than it would for a consultant to become aware of all the implicit and explicit practices of an organisation
- They tend to report to the corporate management. A key lesson of ethnographic studies is that organisations rely on behaviour that would not be found in corporate policy documents. Hence if a consultant is reports to management not users, there is an incentive for them to use the system to impose corporate policy. For the reasons discussed in [DOC REF], this is unlikely to be ideal.

In contrast, end users are members of the community that will use the software. As such they are responsible to their peers, this may help ensure that a balanced solution gets built that doesn't excessively privilege management.

### **5.2.3. Challenges for adoption of End User Security**

So we have seen that of all the people who might define the behaviour of the system, the end users have some strong advantages. However there are a number of social difficulties in how to build end user security systems. Many of these issues would arise with any usable security system, for example: I argued in section [DOC REF] that for a system to be usable to model has to align with the social reality. However social reality may well not match corporate policy. Building a usable security system may involve a negotiation as to how this will be handled. EUD does not create this issue, but it may well bring it to the surface.

Another issue that EUD may highlight is the need to understand the users' domain. It's generally easy to build systems that just present a system's computational abstracts directly on the user interface, and indeed many security systems currently do this. However, to build a framework to support EUD, it's better if these abstractions can more closely align to the users' domain, easing the task of the end user programmers. Again this problem is not new, but EUD highlights it.

Finally there are more cultural issues:

- Security can be viewed as 'too important to be left to amateurs, we need security experts'. However, I have argued that domain knowledge is very important to building usable security systems, so security expertise is not sufficient.
- In some domains end user programmers do not attract the credit they deserve for their work [REF: Segal].
- As discussed in section [DOC REF] security culture is not currently ideal for building end user systems. Hopefully this paper can continue the process of addressing this.

## **5.3. Technical Aspects**

There are also technical challenges in creating EUD systems. The majority of user interaction frameworks are intended to direct manipulation interfaces such as dialogs. As building an end user framework often requires more sophisticated technologies, and as such can require more implementation effort. However, I argue that this is just a symptom of the extra generality and flexibility needed for EUD. As I argued in section [DOC REF], this was going to have to be built to customise the system for individual social environments anyway, so it's probably better to design it in rather than have to retrofit it.

As well as technical difficulties in implementation there are also difficulties in designing the abstractions for a given domain. As Nardi argued [REF], and we expanded upon in section [DOC REF: Comp. determinism], this is important as they can shape the type of programs the end user can construct. Better analytical and investigatory techniques for understanding what these abstractions should be are needed.

However, as we have seen in section [DOC REF: Comp. determinism] failing to select the right abstractions can cause severe usability problems independent of whether we're supporting EUD or not.

### **5.3.1. Negotiable security technology**

As well as the previous general difficulties of building new types of systems, there is a particular challenge for security. As we saw in [DOC REF] systems will need to co-evolve with their users. It is not obvious how to do this in security, which aims to give absolute answers about what behaviour is possible and impossible with a system. As we have seen extensively, if the wrong behaviour is enforced poor usability will result.

An alternative might be to build systems that are relatively 'malleable' at first. A recommended route, as opposed to a mandatory route, is suggested. Other possibilities are possible, but using them is made artificially harder. As the actual working requirement of the system become clearer, perhaps during a trial deployment, it can become progressively harder to perform tasks that 'shouldn't occur'. Eventually a final point may be reached when the system is 'baked', at which point behaviours that shouldn't occur will become impossible and the system is now ready to deal with an environment containing external threats.

Such a system has a number of interesting properties:

- It offers the potential of causing less serious disruption with the deployment of a new security system
- It brings the design of security systems closer to that of 'normal' software engineering, where workarounds would occur with social mechanisms that may be damaged by a 'hard' security system
- At the point of 'baking' the system, it may become clear that some of the 'normal activities' represent security threats. However, this will often not be a failure of system, but rather it revealing a hidden practice which existed anyway. This practice can then be addressed through normal risk management procedures

Such a system would require substantial technical innovation to implement. Further, such a system will present new usability challenges. The final point to be made is that adopting the view of security systems as programming (as we argued in [DOC REF]), allows us to adopt powerful tools from HCI.

### 5.3.2. Theoretical Generality

Much of the work in HCI-SEC so far has generated adhoc heuristics for building better interfaces [REF: PGP study][REF: Garfinkel PhD]. This bears some similarity to the early work in HCI [REF: HE]. However more recently, the HCI community that does research into programming language design has developed a number of analytical tools for discussing the usability of programming languages. I argue that these analytical tools can encompass and generalise many of the heuristics found in HCI-SEC.

Amongst the most powerful of the modern analysis techniques is the Cognitive Dimensions (CDs) of Notations framework [REF]. Whilst the detail is far beyond the scope of this paper, the framework considers the usability of an interface along a series of dimensions, for example, *viscosity*<sub>CD</sub>, the resistance to structural change. This is used to highlight areas in which usability problems are likely to occur. To demonstrate that the CDs can express and generalise the heuristics of HCI-SEC, consider the following passage from [REF: Garfinkel]

‘**Consistent** Meaningful Vocabulary: Prevent confusion by using words **consistently** to convey the **same idea or concept** in different programs and contexts. Likewise, prevent confusion by assigning **consistent meanings to the same word** in different applications or contexts.’

(Emphasis added)

This corresponds well in the CDs framework to saying, that *consistency*<sub>CD</sub> is helpful, as is a good *closeness of mapping*<sub>CD</sub>. Likewise:

‘Delayed **Unrecoverable** Action: Give users **a chance to change their minds** after executing an **unrecoverable** action’

(Emphasis added)

Can be interpreted as arguing that *premature commitment*<sub>CD</sub> is harmful. And finally

‘Complete Delete: Ensure that when the user deletes the **visible representation** of something, the **hidden representations** are deleted as well.’

(Emphasis added)

Is suggesting that *hidden dependencies*<sub>CD</sub> are harmful. Good role *expressiveness*<sub>CD</sub> and *visibility*<sub>CD</sub> are needed.

Such an analysis allows us to produce an analytical framework with which we can view the likely usability properties of a notation in a more systematic manner than is possible with adhoc heuristics. This will allow many of the lessons learnt in programming language design to be applied to security in an accessible manner. Further work in this area is needed, but the initial explorations look promising.

## Conclusions

[TODO]

Usability is poor. Mechanism studies can't fix it alone. Ethnography is one solution but has a lot of problems. EUD offers the ability to put the tools into the hands of the people who know the most about the way the world really works. This should be beneficial. Also, viewing the world from the EUD perspective gives us insights into how to do security usability, such as CDs generalising much of the growing heuristics literature.

## Acknowledgements

Cecily

Much of this work was completed whilst employed by nCipher. Luke's current research is supported by the Kodak Eastman Company.

---