

Usable Morality – A Challenge for End User Security

Luke Church, Alan F. Blackwell

*Computer Laboratory
Cambridge University
luke@church.name, Alan.Blackwell@cl.cam.ac.uk*

Keywords: POP-II.B. Design; POP-IV.A. Top-Down; POP-IV.B. User Interfaces; POP-IV.B. Security;

Abstract

Security policies are even harder to write than other system requirements, because they depend on the description of moral judgments rather than of specific behaviours. Interpreting moral judgments in the context of actual software operations is not feasible for computers, but is also too much like programming to be unproblematically delegated to end-users.

Security Policies

The top down approach to security engineering often prescribes a pattern along the lines that the analyst defines a ‘Threat Model’, leading to a ‘Security Policy’, and then to the deployment of ‘Security Mechanisms’. The security policy phase, in the context of a top-down security engineering process, is intended to provide a summary of what the security mechanisms are intended to achieve. For example such a policy might include ‘The user may play the music file a maximum of three times’ or ‘only members of the Accounts Group may alter the accounts spreadsheet’.

However, businesses experience substantial difficulties in crafting such policies. Anderson (2001) describes a typical corporate security policy as ‘a collection of vapid statements.... the term ‘security policy’ is widely abused to mean a collection of managerialist platitudes’.

Security models are used as part of the process defining a security policy. One of the best known of these models the Bell-LaPadula (1973) Model. In informal terms this defines two levels, high and low, and two rules:

- No Read Up: No process may read data at a higher level
- No Write Down: No process may write data to a lower level

This is a very simple security model and yet has experienced huge difficulties in practical implementations, ranging from failure to account for covert channels (signalling down through unanticipated channels) to mismatches between the simple model and the complex working practices of the organisation. There have also been serious difficulties reasoning mathematically about security policies, resulting in much effort by the formal methods community (MacKenzie 2001). As an indicator of the complexity that might arise in a large system, a security model for the Electronic Health Record of the UK’s NHS IT project consists of some 375 formal rules (Mortiz 2005)

End User Security Policies

It is clearly an unreasonable expectation that end-users might engage in such processes to manage, for example, the security of their home PC. However there is a further more general trend to be established here. Developing a *security policy is a form of programming* and one which exhibits substantial intrinsic complexity. In contrast, human expectations of secure system behaviour are expressed in terms of general aspects of human experience, rather than the detailed specifics of machine operations. This is equally true outside the domain of security, with conventional software

requirement documents, which often describe the objectives and intentions of the system in terms that are not easily interpreted at the level of machine abstractions (and may even be ambiguous, or dependent on social context and interpretation). These factors are true to at least the same degree when the requirement is no longer a concrete behaviour in the world being modelled, but rather an abstract description of the possible worlds in which security violations might occur. As the repeated failure of the waterfall model shows, it is unrealistic to expect even experts to be able to develop a correct policy on the first attempt, let alone an end user.

Morality as a Security Policy Surrogate

Given that the standard top-down approach to security engineering is so inappropriate for end users, the user must instead rely on an approximate understanding of the desired security behaviour. This is often expressed in moralistic terminology, a trend which is encouraged by corporate marketing of security such as: [emphasis added]

‘The Mac OS X default configuration, in contrast [To MS Windows], **guards against shady characters** who could so easily **take control** of your system’ (Apple 2006)

In the absence of a formal security model, the behaviour of a security application, such as a malware scanner (e.g. virus detector) is to establish whether a user would consider a program to be ‘good’ or ‘bad’. Considering a malware scanner from this perspective leads to an analogy with the legal system.

A piece of software is by default assumed to be (good:software::innocent:legal), it is (evaluated::tried), including checking against a list of known malware (signature detection::archive of suspect DNA) and if not permitted, then isolated (quarantined::imprisoned) or removed (deletion::execution).

The Usability of Morality

However such signature based detection systems, the mainstay of virus checkers, have many technical problems and apply poorly to new generation security problems faced by end-users, such as phishing.

The design problem for the developer of security products then becomes: ‘given a set of technical evidence about the behaviour of a piece of software, how can we prompt the user to tell us which moral category that software falls into (good/bad or fully trusted/partially trusted/untrusted)’ Furthermore, the developer must determine: ‘how should our security system respond, based on what the user has told us?’

There are substantial difficulties here, the evidence will be highly technical, our analogy with the legal system would imply that expert witnesses are needed to ‘explain any facts that may not be within the common experience of the jurors’ (Allen 2004). However this may well encompass most of the evidence.

Further, steps have to be taken to ensure that the attacker cannot use this evaluation process to manipulate the *cognitive channels* present in the interface, or else new usability vulnerabilities are likely to be exposed (Church 2006)

There are some indications that this problem is starting to be addressed in simple cases, for example in the way that Microsoft Vista responds to certificate problems with a website. However there is also evidence (Whitaker 2006 Personal communications), that problems arising from the Usability of Morality have prevented several security products being brought to market.

The impact of the Usable Morality now stretches back into the legal domain with the recent FTC ruling against Zango (FTC 2006), which deemed that Zango ‘made identifying, locating, and removing their adware extremely difficult for consumers’ and ‘represented to consumers that the adware did not show popup ads and/or exaggerated the consequences of uninstalling the adware’, making it harder for the user to make their decision about the software’s morality.

References

- Allen, C. (2004) Practical Guide to Evidence, Cavendish Publishing
- Anderson, R. (2001) Security Engineering: A Guide to Building Dependable Distributed Systems, Wiley
- Apple (2006) <http://www.apple.com/macosx/features/security/>
- Bell D.E., LaPadula L.J. (1973) Secure Computer Systems: Mathematical Foundations
- Church, L. (2006) Security Usability, PPIG WIP Workshop
- Federal Trade Commission (2006) In the Matter of Zango, Inc. FTC File No. 052 3130
- MacKenzie (2001) Mechanizing Proof, Computing, Risk, and Trust, MIT Press
- Mortiz, 2005 Cassandra: flexible trust management and its application to electronic health records, University of Cambridge Computing Laboratory Technical Report, TR-628