The Initial Costs and Maintenance Costs of Protocols

Ross Anderson

University of Cambridge

Software-engineering academics focussed for many years on the costs of developing the first version of a product, and ignored the costs of subsequent maintenance. We taught our students the 'waterfall model', and biased research towards the sort of tools and ideas that complemented it, such as formal methods. Meanwhile the economics of software had changed. Software is now so complex that the only way to build version N is to start with version N-1. Iterative development methodologies now rule, and the tools that real developers say have helped them most in the last fifteen years are not theorem provers, but automated regression-testing and bug-reporting systems. Nowadays, the maintenance is the product.

Security engineers have been falling into a similar trap. For years, we thought that the problem of authentication began and ended with trustworthy bootstrapping. Once Alice and Bob shared that elusive session key – and could prove mathematically that no-one else did – we could type up the research paper and head for the pub. Again, the real world has changed. Security maintainability is the elephant in the living room; people know there's an awful problem but are generally too polite to mention it (especially as we don't really know what to do with the beast). Vendors used to not care very much; after all, people replace their mobile phones every year, and their PCs every three to five years, so why not just wait for the vulnerable equipment to be thrown on the skip? With luck, vulnerability scares might even help stoke the upgrade cycle.

But attitudes are changing. The hassles caused by vulnerable machines (both directly and indirectly) continue to grow, and consumer expectations harden. Meanwhile, all sorts of consumer durables are acquiring CPUs and communications. If an airconditioner turns out to have a stack overflow in its TCP/IP code, how do you patch it? If you don't, then how do you deal with a virus that switches millions of airconditoners on and off simultaneously, causing a cascade failure of the power grid? And even before we get to the nirvana of pervasive computing, the economics of patching ordinary PCs has become a large and growing topic in security economics.

A number of ideas have emerged recently about designing protocols for maintainability. In [1], for example, we explored what happens when a principal deploys 'smart dust' in an area that is shortly afterwards attacked by an opponent. Assuming that ultra-low-cost dust motes cannot be made tamper-resistant, the opponent can recover shared secrets by reverse engineering a handful of motes and can then eavesdrop on any links whose communications she happens to have

monitored. This turns out to be equivalent, in some sense, to a network whose nodes must send initial key material to each other in the clear. Does this make security impossible? Not at all. Under assumptions that are reasonable in many applications of interest, the opponent will compromise only a small proportion of the links, and this proportion can be kept both low and stable by combining such techniques as key updating and multipath key combination.

A similar situation arises in peer-to-peer systems, where one might assume that principals start out honest but that a small proportion of them may be subverted once the network becomes busy. Here, the problem is not initial key setup, but the design of mechanisms that limit the damage that a compromised node can wreak.

A third type of problem is rate limiting. Many attacks nowadays consist of the industrial-scale repetition of acts that are, on a small scale, not only harmless but encouraged. Examples include downloading a web page, opening an email account and even just sending an email. Here it is not the act that matters, so much as the aggregate, and much of the mischief is perpetrated by machines that started out honest but were subverted along the way.

Economics can give us some insight into how we should analyse and prioritise defence [2]. In the case of smart dust, for example, the game of attack and defence will depend on both the initial and marginal costs, of both the attacker and the defender. Equilibrium depends on marginal costs – attacker efforts versus defender resilience – and the usual outcome will be either that the attacker gives up, or the defender has to go all out to maintain his network. In many cases, it will be rational for the defender to invest in resilience, rather than in more secure bootstrapping. This is yet another reason why shipping an insecure system and patching it later may be rational economic behaviour (in addition to the market races that already provide one such reason in the case of industries with strong first-mover advantages, such as software [3]).

In the specific case of security protocols, there are other difficult issues. Protocol upgrades can be extremely expensive – a change to a bank card payment protocol, for example, can involve 20,000 financial institutions and dozens of software vendors. The design work for a protocol upgrade can be fraught: failures often occur when so many features are inserted that they start interacting, and from multiple backwards-compatibility modes [4]. And that's in the simple case, where the principals agree on the protocol's goals! Where some principals' interests conflict with others' interests – as happens with applications from privacy enhancement through accessory control to DRM – the protocol specification itself becomes the battlefield. How can we analyse all this?

Real-world failures of security protocols may sometimes be explained by economics. Protocols are somewhat like infrastructure, yet are generally not regulated by any government (they change too quickly). They also don't, in general, make their owner monopoly profits: recent attempts by Microsoft to capture the spillover from their server-side networking protocols have been firmly rebuffed by

the competition authorities in Europe. So it's unclear that any single principal has sufficient incentive to undertake all the work (and liability) of protocol maintenance. And if one were to hope that collective effort might suffice, then beware the tragedy of the commons: each developer will add features to help its clients and ignore the side-effects on other developers and their clients. While a small number of protocol maintainers might conceivably sort out their dependencies by contract, this doesn't help as complexity explodes, as many protocols rely on other protocols, and the whole business gets tied up with business models such as aftermarket control.

At the deepest level, it's unclear what protocols are – or what they will become once computers are embedded invisibly everywhere. Are protocols more like roads, or more like laws, or more like mechanical interfaces, or more like habits? Different stakeholders will have different views, and we are likely to see some interesting tussles.

References

1. "Key Infection: Smart Trust for Smart Dust", Ross Anderson, Haowen Chan and Adrian Perrig, in *Proceedings of the 2004 International Conference on Network Protocols* IEEE October 2004, ISBN 0-7695-2161-4 pp 206–215; at `http://www.cl.cam.ac.uk/users/rja14/Papers/key-infection.pdf`
2. "System Reliability and Free Riding", Hal Varian, at Workshop on Economics and Information Security, 2002; at `http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/49.pdf`
3. "Why Information Security is Hard – An Economic Perspective", Ross Anderson, in *Proceedings of the 17th Computer Security Applications Conference*, IEEE Computer Society Press (2001), ISBN 0-7695-1405-7, pp 358–365; also given as a distinguished lecture at the Symposium on Operating Systems Principles, Banff, October 2001; at `http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf`
4. "API-Level Attacks on Embedded Systems", Ross Anderson, Mike Bond; in *IEEE Computer* v 34 no 10 (October 2001) pp 67-75; at `http://www.cl.cam.ac.uk/users/mkb23/research/API-Attacks.pdf`