DOI:10.1145/1592761.1592773

Butler Lampson

Privacy and Security

Usable Security: How to Get It

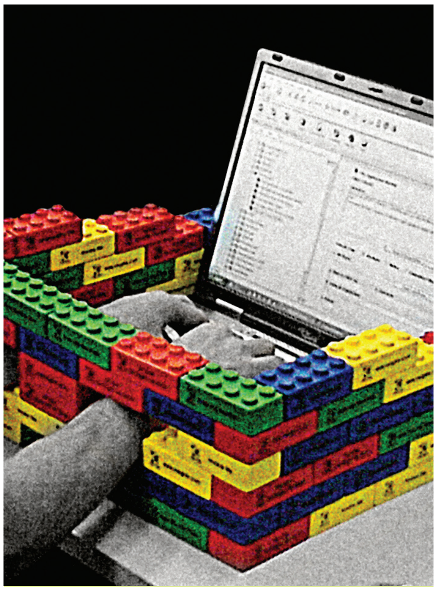
Why does your computer bother you so much about security, but still isn't secure? It's because users don't have a model for security, or a simple way to keep important things safe.

Computer security today is in bad shape: people worry about it a lot and spend a good deal of money on it, but most systems are insecure.

Security is not about perfection. In principle we can make secure software and set it up correctly, but in practice we can't, for two reasons:

▸ *Bugs*: Secure systems are complicated, hence imperfect. Of course software always has bugs, but even worse, security must be set up: user accounts and passwords, access control on resources, and trust relationships between organizations. In a world of legacy systems, networked computers, mobile code, and changing relationships between organizations, setup is error-prone.

▸ *Conflicts*: Even more important, security gets in the way of other things you want. In the words of General B.W. Chidlaw, "If you want security, you must be prepared for inconvenience."[a] For users and administrators, security adds hassle and blocks progress. For software developers, it interferes with features and with time to market.



To make things worse, security is fractal: Each part is as complex as the whole, and there are always more things to worry about. Security experts always have a plausible scenario that demands a new option, and a plausible threat that demands a new defense. There's no resting place on the road to perfection.

Security is really about risk management: balancing the loss from breaches against the costs of security. Unfortunately, both are difficult to measure. Loss is the chance of security breaches times the expense of dealing with them. Cost is partly in dollars budgeted for firewalls, software, and help desks but mostly in the time users spend typing and resetting passwords, responding to warnings, finding workarounds so they can do their jobs, and so forth. Usually all of these factors are unknown, and people seldom even try to estimate them.

More broadly, security is about economics.[2] Users, administrators, organizations, and vendors respond to the incentives they perceive. Users just want to get their work done; they don't have good reasons to value security, and view it as a burden. If it's hard or opaque, they will ignore it or work around it; given today's poor usability they are probably doing the right thing. If you force them, less useful work will get done.[1] Tight security

a Chidlaw, B. Dec. 12, 1954. Quoted by the International Spy Museum, Washington D.C.

PHOTOGRAPH BY IAN LLOYD

usually leads first to paralysis and then to weak security, which no one complains about until there is a crisis.

Administrators want to prevent obvious security breaches, and avoid blame if something does go wrong. Organizations want to manage their risk sensibly, but because they don't know the important parameters they can't make good decisions or explain their policies to users, and tend to oscillate between too much security and too little. They don't measure the cost of the time users spend on security and therefore don't demand usable security. Vendors thus have no incentive to supply it; a vendor's main goal is to avoid bad publicity.

Operationally, security is about *policy* and *isolation*. Policy is the statement of what behavior is allowed: for example, only particular users can approve expense reports for their direct reports or only certain programs should run. Isolation ensures the policy is always applied. Usability is pretty bad for both.

Policy

Policy is what users and administrators see and set. The main reason we don't have usable security is that users don't have a model of security they can understand. Without such a model, the users' view of security is just a matter of learning which buttons to push in some annoying dialog boxes, and it's not surprising they don't take it seriously and can't remember what to do. The most common user model today is "Say OK to any question about security."

What do we want from a user model?

- It has to be simple (with room for elaboration on demand).
- It has to minimize hassle for the user, at least most of the time.
- It has to be true (given some assumptions). It is just as real as the system's code; terms like "user illusion" make as much sense as saying that bytes in RAM are an illusion over the reality of electrons in silicon.
- It does *not* have to reflect the implementation directly, although it does have to map to things the code can deal with.

An example of a successful user model is the desktop, folders, and files of today's client operating systems. Although there is no formal standard for this model, it is clear enough that users can easily move among PC, Macintosh, and Unix systems.

The standard *technical* model for security is the access control model shown in the figure, in which isolation ensures there is no way to get to objects except through channels guarded by policy, which decides what things agents (principals) are allowed to do with objects (resources). Authentication identifies the principal, authorization protects the resource, and auditing records what happens; these are the gold standard for security.[3] Recovery is not shown; it fixes damaged data by some kind of undo, such as restoring an old version.

In most systems the implementation follows this model closely, but it is not very useful for ordinary people: they take isolation for granted, and they don't think in terms of objects or resources. We need models that are good for users, and that can be compiled into access control policy on the underlying objects.

A user model for security deals with policy and history. It has a vocabulary of objects and actions (nouns and verbs) for talking about what happens. History is what *did* happen; it's needed for recovering from past problems and learning how to prevent future ones. Policy is what *should* happen, in the form of some general rules plus a few exceptions. The policy must be small enough that you can easily look at all of it.

Today, we have no adequate user models for security and no clear idea of how to get them. There's not even agreement on whether we can elicit models from what users already know, or need to invent and promote new ones. It will take the combined efforts of security experts, economists, and cognitive scientists to make progress. Here are a few tentative examples of what might work.

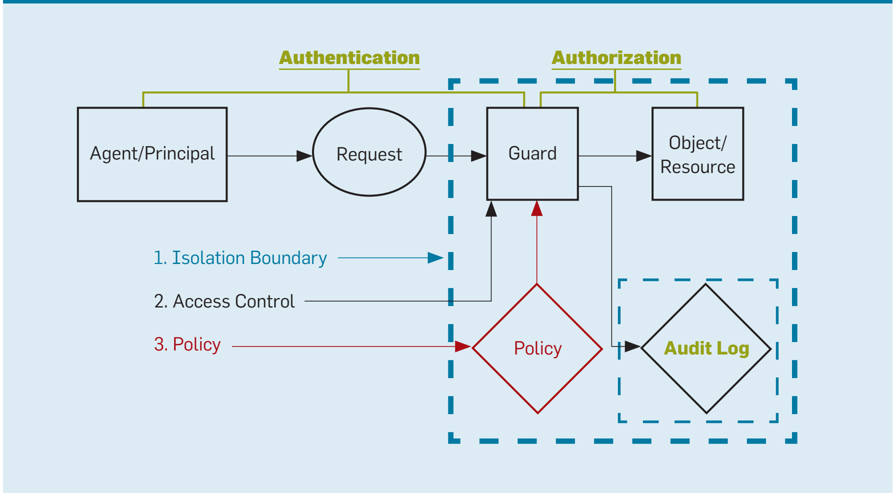
You need to know **who** can do **what** to **which** things. "Who" is a particular person, a group of people like your Facebook friends, anyone with some attribute like "over 13," or any program with some attribute like "approved by Microsoft IT." "What" is an action like read or write. "Which" is everything in a particular place like your public folder, or everything labeled medical stuff (implying that data can be labeled). An administrator also needs declarative policy like, "Any account's owner can transfer cash out."

A time machine lets you recover from damage to your data: you can see what the state was at midnight on any previous day. You can't change the past, but you can copy things from there to the current state just as you can copy things from a backup disk.

Isolation

Perfect isolation ensures that the only way for an input to reach an object is through a channel controlled by policy. Isolation fails when an input has an effect that is not controlled by policy; this is a bug. Some common bugs today are buffer overruns, cross-site scripting, and SQL code injection. Executable inputs like machine instructions or JavaScript are obviously dangerous, but modern HTML is so complex and expressive that there are many ways

Standard technical security access control model.



to trick a browser into running code, and widely used programs with simple inputs like JPEG have had buffer overruns. A modern client OS, together with the many applications that run on it, is bound to have security bugs.

Users can't evaluate these dangers. The only sure way to avoid the effects of dangerous inputs is to reject them. A computer that is not connected to any network rejects all inputs, and is probably secure enough for most purposes. Unfortunately, it isn't very useful. A more plausible approach has two components:

▸ Divide inputs into safe ones, handled by software that you trust to be bug-free (that is, to enforce security policy), and dangerous ones, for which you lack such confidence. Vanilla ANSI text files are probably safe and unfiltered HTML is dangerous; cases in between require judgments that balance risk against inconvenience.

▸ Accept dangerous inputs only from sources that are accountable enough, that is, that can be punished if they misbehave. Then if the input turns out to be harmful, you can take appropriate revenge on its source.

Accountability

People think that security in the real world is based on locks. In fact, real-world security depends mainly on deterrence, and hence on the possibility of punishment. The reason your house is not burgled is not that the burglar can't get through the lock on the front door; rather, it's that the chance of getting caught and sent to jail, while small, is large enough to make burglary uneconomic.

It is difficult to deter attacks on a computer connected to the Internet because it is difficult to find the bad guys. The way to fix this is to communicate only with parties that are accountable, that you can punish. There are many different punishments: money fines, ostracism from some community, firing, jail, and other options. Often it is enough if you can undo an action; this is the financial system's main tool for security.

Some punishments require identifying the responsible party in the physical world, but others do not. For example, to deter spam, reject email unless it is signed by someone you know or comes with "optional postage" in the form of a link certified by a third party you trust, such as Amazon or the U.S. Postal Service; if you click the link, the sender contributes a dollar to a charity.

The most common user model today is "Say OK to any question about security."

The choice of safe inputs and the choice of accountable sources are both made by *your* system, not by any centralized authority. These choices will often depend on information from third parties about identity, reputation, and so forth, but which parties to trust is also your choice. *All trust is local.*

To be practical, accountability needs an ecosystem that makes it easy for senders to become accountable and for receivers to demand it. If there are just two parties they can get to know each other in person and exchange signing keys. Because this doesn't scale, we also need third parties that can certify identities or attributes, as they do today for cryptographic keys. This need not hurt anonymity unduly, since the third parties can preserve it except when there is trouble, or accept bonds posted in anonymous cash.

This scheme is a form of access control: you accept input from me only if I am accountable. There is a big practical difference, though, because accountability is for punishment or undo. Auditing is crucial, to establish a chain of evidence, but very permissive access control is OK because you can deal with misbehavior after the fact rather than preventing it up front.

Freedom

The obvious problem with accountability is that you often want to communicate with parties you don't know much about, such as unknown vendors or gambling sites. To reconcile accountability with the freedom to go anywhere on the Internet, you need two (or more) separate machines: a *green* machine that demands accountability, and a *red* one that does not.

On the green machine you keep important things, such as personal, family and work data, backup files, and so forth. It needs automated management to handle the details of accountability for software and Web sites, but you choose the manager and decide how high to set the bar: like your house, or like a bank vault. Of course the green machine is not perfectly secure—no practical machine can be—but it is far more secure than what you have today.

On the red machine you live wild and free. You don't put anything there that you really care about keeping secret or really don't want to lose. If anything goes wrong, you reset the red machine to some known state.

This scheme has significant unsolved problems. Virtual machines can keep green isolated from red, though there are details to work out. However, we don't know how to give the user some control over the flow of information between green and red without losing too much security.

Conclusion

Things are so bad for usable security that we need to give up on perfection and focus on essentials. The root cause of the problem is economics: we don't know the costs either of getting security or of not having it, so users quite rationally don't care much about it. Therefore, vendors have no incentive to make security usable.

To fix this we need to measure the cost of security, and especially the time users spend on it. We need simple models of security that users can understand. To make systems trustworthy we need accountability, and to preserve freedom we need separate green and red machines that protect things you really care about from the wild Internet. C

References

1. Adams, A. and Sasse, A. Users are not the enemy. *Commun. ACM 42*, 12 (Dec. 1999), 41–46.
2. Anderson, R. Economics and Security Resource Page; http://www.cl.cam.ac.uk/~rja14/econsec.html
3. Lampson, B. Practical principles for computer security. In *Software System Reliability and Security*, Broy et al., Eds., IOS Press, 2007, 151–195.

Butler Lampson (Butler.Lampson@microsoft.com) is a Technical Fellow at Microsoft Research and is an ACM Fellow.

Copyright held by author.