Choose the Red Pill *and* the Blue Pill

A Position Paper

Ben Laurie
Google, Inc.
benl@google.com

Abe Singer
Laser Interferometer Gravitational Wave Observatory
California Institute of Technology
abe@caltech.edu

ABSTRACT

In the movie "The Matrix," our hero Neo must choose between taking the Blue Pill and continuing to live in an online, synthesized fantasy world, or taking the Red Pill and joining the real world. The fantasy world appears to those living in it to be full of flowers and trees and big steak dinners, but unknown to them contains malicious Agents who can alter any portion of the world to suit their needs. The real world, in turn, while real, has no visible sun, and the people have only gray mush for food.

Authorization and authentication of online transactions across a network requires a trusted path between the user and the server. We posit that those who attempt to solve this problem by creating the trusted path on the general-purpose operating system have taken the Blue Pill and are living in a fantasy world. **One simply cannot properly secure a general-purpose operating system**.

Solving the problem by taking the Red Pill and completely replacing currently used operating systems with ones that we can properly secure does not seem palatable. We suggest a solution that involves taking both the Blue Pill and the Red Pill: **providing the trusted path by means of a separate device with a secure operating system, used *in tandem* with the existing general purpose operating system**. Most user interaction occurs on the untrusted system, with the secure device only being used to finalise transactions.

We believe that the technology required for such a device is readily available. Obviously our idea is not a completely novel idea; prior work in the area has had a similar goal. However, most of those attempts have not properly addressed the requirements for the trusted system, generally preferring to use existing general-purpose systems even when on a "dedicated device." [Balfanz 1999] [Kingpin 2001] Others have a very limited scope of use. [Blakely 2004]

We identify a minimum set of requirements for the trusted device. This paper does not provide a working solution (it is a position paper after all); we simply define how one should approach that working solution. Because we advocate a hybrid system it is possible to simplify the trusted system to a point where it would not be usable as a general purpose system, which should make the trusted system rather easier to build and have confidence in.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection – *Access controls, Authentication, Cryptographic controls, Information flow controls, Security Kernels.*

General Terms

Design, Security, Human Factors.

Keywords

Authentication, Authorization, Trusted Path, Red Pill, Blue Pill, Nebuchadnezzar, Scooby Doo, Rotating Shield Harmonics, Secure Operating System, Grey Goo, The Matrix.

1. THE BLUE PILL

If we take the blue pill and continue to live in the reasonably comfortable world of modern operating systems, then we will have a pleasant experience, but we will not be secure. In particular, even if secure protocols are used to defend data in transit, it can still be stolen or manipulated by malware on the machine. **Secure protocols cannot protect against a compromised endpoint.** Solutions which attempt to address compromises on the endpoint merely push the problem around the problem space, and require faith that an attacker has not compromised a particular component of the endpoint.

It is our view that it is infeasible to secure an operating system sufficiently to counter this threat without also making it quite unpleasant to use: it will be inflexible and the user interface will be boring and clunky.

1.1 The Compromised Endpoint Problem

Increasingly, attackers have focused on compromising client endpoint systems, as the average user, not being a professional system or security administrator, does not have the required skills to secure his computer. And we should not expect the average user to have or acquire such skills. By analogy, we would not expect the

NSPW'08, September 22–25, 2008, Lake Tahoe, California, USA.
Copyright 2008 ACM 978-1-60558-341-9/08/09...$5.00.

average car user to perform skilled car maintenance in order to drive to the grocery store.

We assert that all attempts at securing online services eventually fail because of a fundamental failure – extending trust to a system that is untrustable, by attempting to execute trusted transactions on demonstrably untrustable general purpose operating systems. **Nothing can protect software layered upon these operating systems from subversion.**

The "compromised endpoint problem" is not new, but there is clearly no mechanism that allows for transactions to be safely carried out between two endpoints when one of the endpoints is compromised. We assert that no mechanism can be constructed which can make existing clients trustable – the general purpose operating systems in use today are simply do not meet the design criteria for a trustable system and cannot provide a trusted path. Nor do we believe that general-purpose operating systems can ever solve this problem; their attack surface is just too large, and must be so for them to be both useful and usable.

1.2 Rotating the Shield Harmonics

Online services increasingly try to add security "features" to their services to combat compromises. For example, some banks have begun requiring their customers to authenticate with token-based authentication. Other examples include transport encryption, authentication "improvements" such as smart-cards and CAPTCHAs, "trusted images" (where the server shows the user a picture he chose earlier, and expects the user not to log in unless he sees this picture) and system protection schemes such as network firewalls, anti-virus software and "secure" plug-ins for web browsers. [Langweg 2004] [Mannon 2006] [Chiasson 2006]

We refer to this approach of layering/changing security features "rotating the shield harmonics," as the response only stops the attack until the enemy determines the new shield frequency. [STTNG 1990] These approaches only postpone the inevitable, as attackers eventually change tactics to defeat these new features. For example, one attacker's response to token authentication is to compromise the client system, wait for the user to authenticate and then hijack the session post-authentication. Another is to persuade the user to authenticate to a phishing site and then use the token authentication at the real site. [Schneier 2005] Several CAPTCHA systems have recently been compromised. [Protalinski 2008] [Mori 2003] And usability studies show that with the "trusted image technique," one can easily fool the users into ignoring the absence of their picture. [Chiasson 2006]

These shield harmonic rotation schemes all fail because they rely on one or more components of the operating system to not be compromised, yet there is no way to provide that assurance.

1.3 Scooby-Doo Security

Furthermore, many proposed security solutions rely upon the user having in-depth of knowledge of security, behaving properly (every time) and having good judgment. The user is expected to check the validity of certificates, maintain patches and protective software on their system and, most amusingly, not visit "bad" web sites, open "suspicious" attachments, or download and install insecure or malicious software. And, more fundamentally, the user cannot make mistakes.

Unfortunately, for a sufficiently large population, the Bell Curve always applies – some users will simply not get it. [Whitten 1999] A trustable system that can be used by the general population must impose a very low burden of knowledge on the user. Simply expecting users to properly secure their computers and fully understand the threats it is under is an impossible goal. We refer to this ideology as The Scooby Doo approach to security, as failure often involves the designer saying "It would have worked if it hadn't've been for those pesky users." [Scooby 1969]

2. WHAT IS THE TRUSTED PATH?

Much of what is done on a computer is of no immediate security consequence. For example, if we are shopping at Amazon, the process of choosing things to put in our shopping basket does not raise any security issues – we do not need the server to be sure the things it is putting in the basket are the things we intended to be put there. In other words, we can use a "path" from the user to the server that is untrusted.

Until we come to the moment of *paying* for all that stuff, and *choosing where it is delivered*, we can continue to use this untrusted path. However, once we pay and choose a delivery address, we want to be very sure that we are paying for what we wanted and that it is going to be sent where we want. At this moment we need the **trusted path** from the user to the server. This path can take many forms, but the characteristics we want are that the user can be sure they are talking to the correct server, can see what that server intends them to see, that the server can be sure that the user sees what it wants shown, and that the actions taken by the user are faithfully reflected to the server.

2.1 Who Needs the Trusted Path?

Many Internet or network transactions require authentication and authorization. Examples of such transactions include online banking, purchasing merchandise, accessing email, and editing a blog entry. The online service requires authentication and authorization to ensure that only authorized users can access the provided services, and so that one user cannot access the services provided to another user. Users, in turn, want strong authentication and authorization for assurance that transactions they authorize match the transactions they requested. For example, if Neo purchases a book from an online merchant for $50, Neo does not want to later discover that he was actually charged $500 for other products that he did not order and were delivered to someone else. These activities all have a component which requires a trusted path.

Online services increasingly find themselves subject to compromise of user accounts, resulting in theft, fraud, and other malicious activity (e.g. falsifying a blog entry, domain name hijacking, spamming in the user's name and so forth). The user account compromise may stem from a compromise of the server, interception/man-in-the-middle attacks between client and server, or compromise of the client. Increasingly, attackers have focused on the latter of these methods because, as stated above, the average user does not know how to secure his computer. Nor should we expect him to know.

These attacks can result in the theft of credentials which then get used for fraudulent or unauthorized activities. For example, stealing a credit card number, or using an online banking account to transfer money elsewhere. Attackers often leverage credential theft attacks to further compromise other systems. [Singer 2005]

As these attacks increase, users lose confidence in their ability to safely use these services, especially as the perception of the frequency of grows from the possible to the very probable. For example, Trinity will likely stop using online services if she finds that half the time the activity results in fraudulent charges, or some similar difficulty from which she must then spend time to recover from (if possible).

We also anticipate attacks where the transaction shown to user does not represent the transaction actually executed. Restating the example above, Agent Smith has compromised Trinity's machine, and whenever she tries to go to AmaZion, Agent Smith intercepts her web pages and shows her whatever he wants her to see (a man-in-the-middle attack). He delivers a legitimate looking AmaZion page to Trinity's web browser, allowing her to log in and purchase items. So Trinity thinks she has logged in to AmaZion and has authorized a small purchase of books. But in turn Agent Smith gives AmaZion a completely different request for computer parts, which AmaZion believes placed by Trinity. She has no way of determining that the attack has taken place until she sees her credit card bill, and AmaZion has no way of knowing about the attack until Trinity notifies them. Trinity then has the burden of arguing with AmaZion, and her credit card company, over the validity of the transaction, and runs the risk that AmaZion could claim that since the transaction was authorized from her computer, and using her password, she must have authorized the purchase.

3. THE RED PILL

As we have discussed above, taking the red pill to solve our problems may well improve security, but it will vastly reduce usability and fun. The requirement for security will heavily constrain user interface, data sharing and the functionality of software. Much of the software we are used to would be impossible to run on a secure system – for example, web browsers (and even worse, web browser plugins and programming languages – no more Flash, Javascript or PDFs for you).

3.1 Ms. Square Peg, meet Mr. Round Hole

For most things that the user does, he does not need a "secure" operating system. Playing (non-online) games, editing documents, reading the news, listening to music, editing a photo, do not normally require any trusted path operations. Why subject the user to the overhead and inconvenience of a "secured" operating system for these activities?

4. TAKE BOTH PILLS

Our position is that the general-purpose operating system is fundamentally inadequate for trusted operations. One can have a general-purpose system or a trusted system, but one cannot get both in a single package. So two systems are needed.

We propose using the general-purpose operating system for everything but the bits that need security. Have a second system with a built-secure operating system, which operates in tandem with the first. The separate device is built for the purpose of providing a trusted path, and providing a usable interface. We call this device "The Nebuchadnezzar." [Wikipedia 2008] This device is not intended to replace existing systems, but to work in concert with them, being used only for the purpose of handling trusted path activities such as authenticating and authorizing transactions.

This separate system has to be designed to meet the requirements necessary for such trust. The Nebuchadnezzar has two key features: a secure operating system which completely isolates applications from each other, and a user interface which presents the user with information on the transaction being performed (and the application performing it). The secure OS will be "boring" by design, as it will have the capabilities it needs for operation and no more – no flash animation, no paper clip assistants, no minesweeper, skins or wallpaper, no umpteen megapixel camera nor mp3 player. The Nebuchadnezzar should never be considered useful for activities other than trusted path operations.

We envision the Nebuchadnezzar as a small handheld device, with a graphic user interface, manual input (e.g. buttons or touch screen), one or more communications interfaces, such as USB, Ethernet, Bluetooth, etc., and some non-volatile storage. The device may have a unique identifier which a remote system can use to assure itself it is communicating with the same device – or other means might be used, such as a per-application, per-device key (which would have better privacy properties). The device can run multiple applications for different types of transactions, but the server can readily identify which application is being used (for example, by virtue of a secret shared with only that application).

When a transaction between user and a system requires the trusted path, the user can provide the system in touch with the Nebuchadnezzar. The device provides the user with a display of the transaction, and the user then interacts with the device GUI to authorize the transaction.

The communications between the device and the system can provide end-to-end data integrity and confidentiality, so they may utilize an untrusted communication path (e.g. through the general purpose OS, or over an independent wireless connection).

The Nebuchadnezzar is not a one-trick pony, though. We can foresee a range of capabilities for this device, from simple authentication and authorization, to more complex activities such as online banking, shopping or peer-to-peer transactions. In corporate environments such devices could be useful for single sign-on, for maintaining required audit trails and for activities such as delegation of authority.

A net benefit of such a device is that the control of the trusted activity can be placed literally in the user's hands, as opposed to the current paradigm where the operating system and remote site maintain much of the control. For example, currently when a user purchases goods online, the user provides his credit card information to the merchant via a web browser. The user must trust that the information shown on the web page accurately reflects the purchase being performed, and will not know otherwise until the credit card bill arrives. Furthermore, the user must trust that the remote site to whom he is providing his credit card is actually the intended site and that he has not been diverted by an attacker to a malicious site.

With our proposed device, the user shops with their computer as usual, and then, when ready to purchase, the device can present the user with information on the requested transaction, e.g. a list of items to purchase and a dollar amount. The user can then use the device to authorize the transaction. The authorization response is sent over the trusted path. The user has verified the transaction as requested, and the merchant has a trustable signature on the transaction request.

Most or all of the technology necessary to build the Nebuchadnezzar already exists. In this respect we are not trying to invent a new technology, just apply previous work that has gone unused in this context.

Limiting the purpose of the Neb to only trusted path operations allows us to greatly simplify the design of the system, which in turn leads us to believe that building the device would be feasible and cost effective. For example, the applications on the device never have to communicate with each other or share data, so by not having IPC and by partitioning storage in the operating system, we can make it impossible for applications to do so. Such an approach would not be desirable for a general-purpose operating system.

4.1 This Part is Important!

Approaches have been proposed which, at least superficially, seem very similar to the Neb. However, they often miss the mark by making use of an insecure operating system, relying on just the fact that the operating system is (somewhat) different. The secure device really needs to be secure; it's not enough to just have a second system that's equally insecure. Thus we have to use an OS on the device that has the appropriate security properties.

Other approaches try to address the secure OS requirement, but do not have enough of a user interface, so the user cannot be assured as to what the device is actually doing. The trusted system has to provide the user with enough feedback that the user can properly understand the transactions he is being asked to authorise. Thus the device has to have an acceptably rich user interface,

We provide detailed requirements below. We emphasize here the key points of our requirements:

1. A non-spoofable user interface so the user knows what the device is doing
2. An operating system that's built secure from the ground up to be secure
3. The device is not intended to be general purpose (e.g. it doesn't run a web browser)

5. USE CASES

We provide here some examples of how different sorts of transactions would play out with the Neb, including how this method compares to similar operations on an insecure operating system.

5.1 Case A: Simple Authentication

Neo wants to edit his blog, which requires that he authenticate to the Zion blog server. He connects to the blog server and makes the required edits (for this example, we assume that no secret information is displayed during the editing process, and therefore no authentication or authorization is needed to start the edit). Once he is happy with his edits, he submits them to the server. The server then establishes a connection with Neo's Neb (it knows which one is his from prior association with Neo's account) and sends it the proposed edit. The Neb displays the edit to Neo and asks him to confirm. When he does, the Neb responds to the server with, say, a signature for the edit. If this matches, the edit is applied. If it does not, then Agent Smith is playing games again and the edit is ignored.

Reviewers have commented that devices like token-based one-time passwords, or sending a one-time password via SMS to a cell-phone, etc. would provide equivalent security. We disagree. Some of those solutions require entering the password via the web browser on the untrusted system. All of them require approval of the edit on the untrusted system. Agent Smith could present Neo with a forged web site, and forged responses from the server, tricking Neo into thinking he is communicating with the actual blog while Agent Smith uses Neo's credential for other purposes (such as writing a blog entry supporting Smith's cause). Neo has no ability to determine what actually happened when he interacted with the server.

Reviewers have strongly argued for the effectiveness of using a cell phone as an out-of-band communications channel. We respond that this approach really just rotates the shield harmonics. The supporters first presume that the cell phone cannot be compromised, whilst the media clearly contradicts this notion. The follow-on presumption is that even if the phone were to be compromised, the attacker would not be able to tie the activity on the phone with activity on the computer. Given that users routinely connect their cell phone and computer together, the ability of an attacker to identify the pairing does not seem so far-fetched.

Should Agent Smith try to use Neo's Neb to authenticate to a different service, Neo will see the request on the Neb, but know that he did not make the request, and reject it. Likewise, if Neo provides an incorrect Neb, he will never see the follow-up on his Neb. In both cases, improper authentication attempts fail.

5.2 Case B: Online Shopping

Neo wants to buy Trinity some flowers. Neo shops online with his web browser at AmaZion.com, and at checkout AmaZion connects to his Neb. AmaZion sends the purchase information to the Neb. The AmaZion application on the Neb presents Neo with the order and the purchase amount. Neo can verify these and authorize the transaction with his private key. Should Agent Smith try to spoof Neo's web browser and try to purchase $1000 of software, Neo will readily see that the transaction amount on the Neb does not match what he sees on the screen, and refuse to authorize the sale. Furthermore, the Neb could store the signed authorization as a purchase receipt.

Should Agent Smith spoof Neo's browser and try to make purchases at AmaZion with Neo's account, Neo will see that the authorisation request on his Neb contains the wrong items, and can refuse to authorize the request.

Similarly, if Smith uses his own machine to try to simply impersonate Neo at AmaZion it will all go well until he clicks the "buy" button – at which time, at worst, Neo's Neb will unexpectedly ask him to authorise a purchase, which he will decline. At best, the Neb won't be online, so Neo won't even be bothered.

6. PRIVACY

Although we do not specifically discuss privacy in most of this paper, it should be noted that The Neb is neutral from a privacy perspective. Because each user/server pair ends up with its own keys, by necessity, the Neb reveals no more about its user than would be revealed in any case (i.e. the linked history of all his actions at each particular site, not linked between sites). It should also be possible for a user to have multiple independent accounts at the same server without The Neb giving the game away.

Of course, a Neb-like device could choose to link the user across sites, but it is by no means required by the design.

7. REQUIREMENTS

As we stated above, achieving an actual trusted path requires specifying (and implementing) proper requirements for the device. Here we elaborate on the minimal set of requirements for the Nebuchadnezzar to fulfill its role.

- It must be able to do cryptography.

This is needed because the Nebuchadnezzar will have to prove that it was involved in transactions. As far as we know, the only efficient and practical way to provide such proofs is through cryptography.

- It must be able to do asymmetric cryptography.

Although it is possible to use symmetric ciphers or hashing to prove involvement, this always leaves one side open to the other party in the transaction forging the transaction. The only way for Neo to prove that he and only he (or rather, his Nebuchadnezzar) was involved is by doing asymmetric cryptography.

- It must interact with the user's untrusted system.

It would be nice to demand that users only ever used their trusted system to do anything leading to a transaction requiring trust. This is not realistic. One can hardly expect the user to do all their browsing on their Nebuchadnezzar, for example (indeed, if that were possible, then their untrusted system could be trusted, but we claim that cannot be so).

Therefore we must allow the user to do most of the work on an untrusted system and only involve the trusted system at the moment of trust.

- It must have a user interface.

It is tempting to suppose that we could allow the UI to take place on the untrusted system and only delegate cryptographic operations to the trusted system. But then we have a problem: the untrusted system could be showing "authorise the purchase of this $10 book from Amazon" while the trusted system is signing "give $10,000 to evil.org".

Therefore the trusted system must at least be able to display the thing it is about to sign (or otherwise cryptographically process) and confirm that the user intends this action. Note that this may mean that the trusted system has to be able to interpret and display encoded (as opposed to encrypted) data.

- It must be updateable

It is implausible to think that we could design and ship a device that, without change, can handle all current and future authentication and secrecy requirements. It will also be necessary to update keys for CAs and other trusted third parties.

- It must be able to run multiple applications

It seems equally implausible to suppose that there's a one-size-fits-all application that could meet all our cryptographic needs. For example, the application that authorizes a $10 purchase from Amazon is probably not the same one we want to do our banking with, and neither would be suited to sending encrypted messages to our mistresses.

In particular, it is important that the user be able to clearly understand what he is doing when using the Nebuchadnezzar, and this is likely to require a diversity of user interfaces.

- It must have an absolutely bullet-proof kernel

As will be seen from discussion below, security of the system will not rely on trust in the application but will derive solely from the security of the kernel. Therefore the kernel needs to be secure!

We do not address in detail how this might be done, but we do claim that the substantially reduced complexity of the trusted operating system, which it enjoys precisely because it is not required to be general purpose, will at least reduce the task of providing convincing security from the clearly insurmountable problem of securing, say, Windows.

7.1 What It Doesn't Need

So how does this differ from our standard untrusted operating systems?

Most importantly. it doesn't have to run more than one application at the same time. We are either doing banking or buying a book from Amazon, not both at once. Eliminating multi-tasking eliminates side-channel attacks and wall-banging. It also makes it possible to absolutely partition resources – only those needed by the currently running application need to be available at all.

It doesn't have to support IPC. Obviously, eliminating multitasking takes most of the fun out of IPC, but it is worth stating that there's no need for tasks to communicate. In fact, there's probably no need for tasks to share any resources (including cryptographic keys) at all, other than communications devices. If tasks do not share resources then the threat posed by untrusted programs on one's Nebuchadnezzar is vastly reduced. Indeed, it may even be possible to run arbitrary code on the system since the code would have no access to any resources other than its own: obviously the absence of side-channel attacks is crucial to this argument.

If the code chosen to represent any particular trust relationship is agreed by both parties then it can be argued that it doesn't matter if the code is "malicious": this is what the parties intended (or, at least, agreed to). In any case, maliciousness is restricted to that single relationship, which is a vast improvement over the current status. It is worth noting, however, that insecure applications might lead to malicious attacks that are not the responsibility of either party – for example, if an application can be persuaded to display something other than what was intended, then it could be abused by a third party. So, the presence of a secure operating system does not absolve application developers from their duty of care. But it does at least make it realistically possible for them to discharge that duty.

This leads to another requirement.

- It must be able to attest to the software it is running.

If Neo is the user, and Trinity is the remote half of a transaction the Nebuchadnezzar is participating in, she may want to know that the device is running the software she agreed (with Neo) it should run in order to perform the transaction (for example, if the transaction is legally binding Trinity may want to be able to prove that Neo must have given consent). Trinity may also want to be assured that Neo has been diligent in keeping the device up-to-date.

Note that this requirement gets us on to potentially thin ice with user-unfriendly practices, such as DRM. We would argue that we should not throw the baby out with the bath water. Just because evil is possible with a device, that is not sufficient reason not to use the device: one should, instead, refrain from using and campaign against the evil and benefit from the good uses. This is definitely uncontroversial where the owner of both ends of the connection is the same person.

So can we build this OS? It seems to us that we can, and it must be easier than securing an existing general purpose OS, or creating a new, secure, general purpose OS, though there definitely some research challenges.

8. RELATED WORK

As we mentioned above, attempts have been made to provide solutions similar to what we propose. However, in our view, those solutions did not understand the necessary requirements. We discuss some of those solutions here.

8.1 Smartcards

Whilst smartcards have some of the data and application isolation we require, they have no user interface. In general, they rely on UI displayed on the untrusted system, which clearly can be subverted to show one thing while the smartcard is authorising another.

8.2 Mobile Phones

Mobiles come closer to the Neb's requirements, but still, they tend to be too open. Multitasking is allowed, as are IPC and shared data. They run browsers, too. We do admit that it might be possible, with sufficient care, to somehow share the same device as a mobile phone and as a Neb.

8.3 Operating System Plugins

The claim is sometimes made that an OS plugin could somehow have better security than the rest of the OS and this provide the trusted path and the isolated storage. Although there is some merit to this idea, it is pretty clear that you still have to secure most of the operating system in order to protect this component, and we have already claimed that that is not possible.

8.4 Browser Plugins

Pretty clearly, browser plugins stand even less chance than operating system plugins – firstly, because the browser is, in practice, a monolithic application, it is not possible to secure one component from another, and secondly because any user interface the browser might display can clearly be simulated by any other application program – and likely it can be controlled by other applications, too.

9. OPEN QUESTIONS

- Can we strongly tie the transaction to the user interface?

It is vital to this proposal that the user is shown what they are authorising. This means that there must be a direct link between what is displayed and the underlying cryptographic operation. Can we provide a UI that is sufficiently usable and pleasant whilst maintaining this link?

It seems trivially true that we can supply an unpleasant and unusable UI for this; the question is whether we can improve on that without compromising the security of the device.

- Can we strongly tie the transaction to the work done on the user's (untrusted) machine?

For example, if the user's machine tells him he is doing business with Amazon, and The Neb tells him he is doing business with Amaz0n, will he notice? One obvious idea here is to use the petname system [Miller], but this only solves the problem for domain names. Can we extend petnames to cover other aspects of transactions? For instance, if the transaction is via a third party, say Paypal, then the pet name would cover Paypal itself, but would it naturally extend to cover the eventual recipient of the funds? Likewise, if we were buying a Canon D7000 from eBay, would we notice if the trusted device told us we were buying a Can0n D7ooo?

- How do we persuade users to carry such a device?

It is sometimes suggested that users would be reluctant to carry a device just for the purpose of security, and this has led to many attempts to instead use inherently insecure devices, such as cell phones, for the purposes we describe. However, users seem willing to carry an extra device (e.g. iPod) when they perceive the value of doing so to be high. Our view is that users already carry devices solely intended to provide security benefits, so there is no fundamental problem here. A couple of examples are keys and credit cards. The open question is how to persuade users to view The Neb as something in the same class as the things they already carry.

- What happens when the device is lost?

Like any other credential, loss is bad. Centralised revocation and re-issue seems the way to go here, but it would be fair to say that we do not have real-world evidence that this is a solved problem, despite its apparent tractability.

10. REFERENCES

[AP 2002] "Mobster's son pleads guilty of gambling; computer spying helped seal case," *Associated Press*, March 1, 2002.

[Blakely 2004] Blakely, R. and Blakely, G.R. "All Sail, No Anchor II: Acceptable High-End PKI*," International Journal of Information Security* (2004) 2(2):66-77.

[CERT 2008] "Adobe Flash Updates for Multiple Vulnerabilities*," Technical Cyber Security Alert TA08-100A.* US-CERT. April 9, 2008. Retrieved on April 20, 2008, 18:09 PST. <http://www.us-cert.gov/cas/techalerts/TA08-100A.html>

[Chiasson 2006] Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. "A Usability Study and Critique of Two Password Managers." In *Proceedings of the 15th USENIX Security Symposium*. August 2006.

[Balfanz 1999] Balfanz, D. and Felten, E. W. 1999. "Hand-held computers can be better smart cards." In *Proceedings of the 8th Conference on USENIX Security Symposium*. August 1999.

[Kingpin 2001] Kingpin, K. and Mudge, M. 2001. Security analysis of the palm operating system and its weaknesses against malicious code threats. In *Proceedings of the 10th Conference on USENIX Security Symposium*. August 2001.

[Langweg 2004] Langweg, H. "Building a Trusted Path for Applications Using COTS Components." *NATO Research and Technology Symposium IST-041/RSY-013 "Adaptive Defense in Unclassified Networks",* 19 April, 2004.

[Mannan 2007] Mohammad Mannan, P. C. van Oorschot. "Security and Usability: The Gap in Real-World Online Banking." *In Proceedings of the 2007 New Security Paradigms Workshop*. September 2007.

[Miller] M. Miller; "Lambda for Humans – The Pet Name Markup Language"; http://www.erights.org/elib/capability/pnml.html.

[Mori 2003] Mori, G., Jitendra Malik, J. "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA." In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2003.

[Protalinsk 2008] Protalinski, E. "Gone in 60 seconds: Spambot cracks Live Hotmail CAPTCHA." *Ars Technica*, April 15, 2008 - 09:13AM CT, Retrieved on April 20, 2008, 16:03 PST. <http://arstechnica.com/news.ars/post/20080415-gone-in-60-seconds-spambot-cracks-livehotmail-captcha.html>

[Schneier 2005] Schneier, B. "Two-factor authentication: too little, too late." *Communications of the ACM*, Volume 48, Issue 4. April 2005.

[Scooby 1969] multiple episodes. *Scooby Doo, Where Are You.* CBS Television, 1969-1972.

[Singer 2005] Singer, A. "Tempting Fate," *;login*:, Volume 30, #1, Usenix Association, November 2005.

[Spalka 2001] Spalka, A., Cremers, A.B., Langweg, H.: "The fairy tale of what you see is what you sign: Trojan horse attacks on software for digital signature." *Proceedings of the IFIPWG9.6/11.7 Working Conference, Security and Control of IT in Society-II (SCITS-II*). 2001.

[STTNG 1990] "The Best of Both Worlds, Part I." *Star Trek: The Next Generation*. Paramount Television. June 16, 1990.

[Whitten 1999] Whitten, A., Tygar, J. D. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." *In Proceedings of the 8th USENIX Security Symposium*, August 1999.

[Wikipedia 2008] "List of ships in the Matrix series*." Wikipedia, The Free Encyclopedia.* April 15, 2008, 22:29 UTC. Wikimedia Foundation, Inc. Retrieved on April 20 2008. <http://en.wikipedia.org/w/index.php?title=List_of_ships_in_the_Matrix_series&oldid=205892592>.

[Ye 2002] Ye, Z., Yuan, Y., and Smith, S. "Web Spoofing Revisited: SSL and Beyond." *Department of Computer Science Technical Report TR2002-417*. Dartmouth College. 2002.