Choose the Red Pill *and* the Blue Pill

Position Paper

Ben Laurie

Google, Inc.
benl@google.com

Abe Singer

San Diego Supercomputer Center
University of California at San Diego
abe@sdsc.edu

Abstract

In the movie "The Matrix," our hero Neo must choose between taking the Blue Pill and continuing to live in a online, synthesized fantasy world, or take the Red Pill and join 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, is sunless and the food is a gray mush.

Authorization 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. Solving the problem by taking the Red Pill and completely replacing the operating system with one that can be properly secured 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. We believe that the technology required for such a device is readily available, and define the requirements for such a device.

Obviously this is not a completely novel idea, but previous attempts have not entirely addressed the requirements for the trusted system, instead preferring to use existing general-purpose systems. [Balfanz 1999] [Kingpin 2001]

Introduction

Many Internet transactions require authentication and authorization. Examples of such transactions are online banking, purchasing merchandise, accessing email, and editing a blog entry. The online service requires authentication and authorization to assure that services are only provided to those authorized users, 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 are the transactions they requested. For example, if Alice purchases a book from an online merchant for $50, Alice does not want to later discover that she was actually charged $500 for other products that she did not order and were delivered to someone else.

Online services are increasingly 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). These compromises may be the result of a compromise of the server, interception/man-in-the-middle attacks between client and server, or compromise of the client. Increasingly, attackers are focusing on the latter of these methods as the average user is much less likely to have properly secured their computer than has the online service.

These attacks can result in the theft of credentials which are then used for fraudulent or unauthorized activities. For example, stealing a credit card number, or using an online banking account to transfer money elsewhere. Credential

theft attacks are also used to further compromise other systems. [Singer 2005]

As these attacks continue to increase, users lose confidence in their ability to safely use these services, especially as the perception grows that compromise is more likely than not. For example, Alice will likely stop using online services if she finds that half the time the activity results in fraudulent charges or some similar difficulty.

We also anticipate attacks where the transaction actually being executed is different than the one being shown to the user. Although the current lack of strong authentication makes these kinds of attacks superfluous, the use of stronger means of authentication on existing are likely to lead to such attacks.

Online services increasingly try to add security "features" to their services to combat compromises. For example, some banks are requiring their customers to authenticate with token-based authentication. Other examples are transport encryption, authentication "improvements" such as smart-cards and CAPTCHAs, "trusted images" (where the user is shown a picture he chose earlier, and is expected not to log in unless he sees this picture) and system protection schemes such as network firewalls, anti-virus software, "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 it is only a matter of time before 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," users are easily fooled into ignoring the absence of their picture. [Chiasson 2006]

Furthermore, many proposed security solutions rely upon the user having in-depth of knowledge of security, behaving properly (every time) and having good judgement. 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. 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 "if it hadn't've been for those pesky users I would've gotten away with it." [Scooby 1969]

We assert that all these 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 general purpose operating systems which are demonstrably untrustable. Any software layered upon these operating systems cannot be protected from subversion.

The "compromised endpoint problem" is not new, but there clearly is no mechanism which 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 simply too large, and must be so for them to be both useful and usable.

The Trusted Path Problem

The underlying problem is the lack of a trusted path for authentication and authorization of Internet transactions.

General purpose operating systems in use today are incapable of providing a trusted path. Specifically, this category

of systems include Windows (CardSpace's claims notwithstanding), Unix/Linux, and cellphone/PDAs. Furthermore, online service providers increasingly treat web browsers as an operating system -- as many browsers have the self-contained ability to run applications via Java, Javascript, Flash, etc. [CERT 2008]

The trusted path problem is a result of the *design* of these operating systems and hardware. Some PDAs don't have memory protection. Most of these systems have inadequate privilege separation, either through design or user configuration. For example, Windows home users typically have administrative privileges on their computer for ease of software installation.

The design features of these operating systems make it impossible for the user to differentiate between interacting with the operating system and interacting with an application that *looks* like the operating system, or talking to a remote service versus a site that looks like the remote service (the phishing problem). [Felten 1996] [Ye 2002] Even if the operating system itself has not been compromised, the client application may be. Or the user may simply believe the wrong thing, despite the system's and application's best efforts.

The current operating system culture also relies on a complex chain of trust. The user must trust that the developer of the operating system has designed and implemented the system properly, and then separately and individually trust each piece of application software developed by different vendors, plus the device drivers, the network stack, graphics libraries, the browser -- even the keyboard. [AP 2002] As all of these applications are run in the same privilege space with no isolation, compromise of a single application undermines the integrity of all applications. And the user must then trust that patches are legitimate, and downloaded and installed properly. In the relationship between the user and the remote server, the user must trust that server credentials are not compromised, and when new credentials must be installed on the user system (e.g. new CA root certificate), the user has a bootstrapping problem: tnothing provides a trusted path for downloading and installing the credential. And the trust requirements are reflexive: the server must also trust that the client system has not been compromised.

At the application level, even when a user is knowledgeable, code which performs security tasks such as signing with asymmetric keys cannot be verified, nor can the user assess *what* is being signed at any given time. [Spalka 2001] Equally, the operating system and application software are incapable of determining if an action was actually performed by the user or by a software agent. The net result is the user cannot trust that the transaction he is requesting is the one that will be executed, nor that he is even communicating with the intended service. Likewise, the service cannot trust that a transaction is actually being requested by the authorized user. In the end, user credentials are not trustable by the service.

These operating systems are also generally very complex, consisting of thousands of separate components. This complexity makes provability of pretty much any security property impossible (other than "this system is insecure", which we can prove trivially by example).

Why are we using such insecure operating systems? Because they're relatively easy to use. Indeed, we take the stronger position: easy-to-use general purpose systems *must* be insecure.

Position

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 can't get both in a single package. So one needs two.

We propose that the solution to this problem is to allow users to continue using their general purpose system for general activity, and provide a separate, secure operating system for the trusted path. This separate system should be designed to meet the requirements necessary for such trust, and give the user an interface which presents enough information to assess the legitimacy of authorization requests.

We propose the concept of a separate device, 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.

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. 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 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 in the user's hands. 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. Both the request to authorize and the response are done 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.

Requirements for the Nebuchadnezzar

Now that we've explored the problem space we can define a minimal set of capabilities the Nebuchadnezzar requires 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 Alice to prove that she and only she (or rather, her 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.

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 it is doing, 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 Bob is the user, and Carol 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 Bob) it should run in order to perform the transaction (for example, if the transaction is legally binding Carol may want to be able to prove that Bob must have given consent). Carol may also want to be assured that the Bob has been diligent in keeping it 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? Seems to us that we can.

References

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

[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 Defence 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.

[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>

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

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

[Singer 2005] Singer, A. "Tempting Fate," *;login:*, Volumn 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.