Painless Migration from Passwords to Two Factor Authentication

Ziqing Mao[1], Dinei Florêncio[2] and Cormac Herley[2]

[1] *Facebook*
Palo Alto, CA
zmao@facebook.com

[2] *Microsoft Research*
Redmond, WA
dinei@microsoft.com
cormac@microsoft.com

***Abstract*—In spite of growing frequency and sophistication of attacks two factor authentication schemes have seen very limited adoption in the US, and passwords remain the single factor of authentication for most bank and brokerage accounts. Clearly the cost benefit analysis is not as strongly in favor of two factor as we might imagine. Upgrading from passwords to a two factor authentication system usually involves a large engineering effort, a discontinuity of user experience and a hard key management problem. In this paper we describe a system to convert a legacy password authentication server into a two factor system. The existing password system is untouched, but is cascaded with a new server that verifies possession of a smartphone device. No alteration, patching or updates to the legacy system is necessary. There are now two alternative authentication paths: one using passwords alone, and a second using passwords and possession of the trusted device. The bank can leave the password authentication path available while users migrate to the two factor scheme. Once migration is complete the password-only path can be severed. We have implemented the system and carried out two factor authentication against real accounts at several major banks.**

I. Introduction

In the US only a small minority of banks offer two factor authentication. There are a number of possible reasons for this. But high among them is the fact that changing from a single factor (password) authentication system to a two factor system requires a complete overhaul of the legacy authentication system. This is a painful and costly upgrade for any organization to make. Generally there is no evolutionary path to upgrading the authentication infrastructure: the old system must be pulled out and a new one put in place. There is a hard transition when users are forced to begin using the new system.

Not surprisingly many institutions will avoid and delay this pain if possible. Even if the fraud that results from password stealing is large (see *e.g.* [24]) it represents a known cost while the risk of a major glitch or security incident when doing infrastructure changes is unknown. There is the further risk that two factor authentication might result in customer defections or excessive support calls. There is at least the potential for very significant customer dissatisfaction if a hard transition is forced upon users [10]. Banks are probably following the maxim "if it ain't broke don't fix it." While password authentication may appear "broke" to much of the security community the reluctance to move to two factor schemes suggests that it isn't "broke enough" to justify the pain of the proposed two factor fixes.

The system we describe addresses this problem by offering a painless migration path to two factor authentication. First, an institution can upgrade to two factor authentication without altering its legacy authentication servers: the change is strictly incremental, meaning that while an additional server will be added to their data center (in fact it can also reside outside, see Section IV-C), no alteration of their existing system is necessary. The existing servers stay in place and do not have to be upgraded patched or altered in any way. The additional server requires no access to any data on the legacy server (*e.g.* it doesn't require any knowledge of the passwords or even userIDs). Second, it is possible to have both the legacy password authentication and the two factor systems co-exist; this makes it possible to roll the two factor system out gradually. For example some users can continue to use passwords while others use two factor. Or users can use two factor only when the situation demands it (*e.g.* logging in from an internet kiosk). If there are any problems with the two factor scheme reversion is effortless since the old system is still in place. The two factor scheme can be rolled out to different subsets of users to test their reaction and iron out any difficulties before offering it to everyone.

II. Related Work

The Impostor [20] system of Pashalidis and Mitchell, is a password management system where roaming users can

WIFS'2011, November 29th-December 2nd, 2011, Foz do Iguaçu, Brazil. 978-1-4577-1019-3/11/$26.00 ©2011 IEEE.

access their credentials. Rather than have users authenticate themselves by typing a master password, a challenge response authentication is used. The user is assigned a large string that forms the secret. When requesting access the user is challenged to provide characters from randomly selected positions in the string, and is authenticated only if he responds correctly.

Florêncio and Herley [9] describe another proxy web service that allows users to access web-sites by using a MITM proxy. The user's password is pre-encrypted and carried as a list of one-time passwords. Thus the proxy does not store the passwords, but rather the keys with which the passwords have been pre-encrypted. This however is still a single factor scheme. Further, there is a vulnerability to an attacker who steals one-time passwords. The KLASSP system [11] and the (outdated) trick in [8], by the same authors, tries to circumvent keyloggers, but still use a single factor for sign in.

Balfanz and Felten [7] provide smartcard functionality using a mobile device (a PalmPilot). Using a serial link to a PC the device provided a trusted authentication path. They also first discuss the interesting question of splitting trust between a limited capability trusted device (*e.g.* a smartphone) and a powerful but untrusted one (*e.g.* a PC). There have been several other proposals that follow this trust splitting paradigm and involve using a cellphone, PDA or smart-phone to provide a second factor or an out-of-band channel for authentication. Wu *et al.* [18] sketch an architecture where a proxy stores credentials; the proxy delivers a challenge which must be answered by SMS to authenticate the user. It is thus a single-signon scheme where an out of band channel is used for a second factor. Parno *et al.* [6] describe a smart-phone based system that performs mutual authentication, even when users are confused or make bad decisions. The system protects both against phishing (where the user incorrectly believes he is connected to MyBank) and spyware where keyboard traffic is recorded. The system requires a connection (such as bluetooth) between the trusted device and the machine on which the browser resides. Mannan and van Oorschot [17] describe MP-Auth: another system that uses a trusted mobile device such as a PDA or smart-phone to enter the password. The device encrypts the password using the end server's public key before passing it to the untrusted terminal. MP-Auth also requires a channel (such as bluetooth) between the trusted device the untrusted machine. The public keys of the sites to be accessed are pre-loaded onto the device, which prevents the untrusted machine from mounting a MITM attack.

While other proposals address the question of logging in from untrusted environments we believe ours is the first to convert a legacy password system into a two factor scheme. Previous efforts have focused on protecting the user's long term secret [6], [5], [9] but are still single factor schemes: if the user's secret (password) is discovered the system fails. By contrast, in our scheme discovery of the password does not allow access along the two factor path. Thus a bank that has migrated all of it's users (*i.e.* disabled the single factor path as discussed in Section V) would not be vulnerable even if the original underlying secrets (*i.e.* passwords) were discovered. Second, we do not require any connection between the trusted smart-phone and the untrusted machine.

III. Method

A. Legacy Single Factor Authentication

Clearly there will be large differences between the server arrangements of different organizations depending on the scale of the population served. While generalization is hard the server chain will generally have the components shown in Figure 1. A web-server accesses upper level database and application servers that provide the content and functionality. Below it, generally an SSL accelerator handles the burden of (at least) the asymmetric encryption (and possibly the symmetric encryption also). A firewall protects the data center from various forms of attack. Depending on the scale of the system the web server functions may themselves be broken into smaller components. For example, there might be a server that just handles the authentication. Thus depicting the web server as a single block is not intended to suggest that only a single machine is involved.

The web server handles the POST events that come from login forms submitted by users. When the credentials submitted result in a successful login the web-server communicates with database and application servers that retrieve user information and perform whatever functions are supported. At a lower level the traffic may flow through other servers such as an SSL accelerator and a firewall server.

Instead of replacing the authentication server we add another server which we call a Cascader. None of the existing applications on any of the other servers change (see Section IV-C). Thus the system involves merely adding a server and not altering legacy systems. The Cascader sits between the upper (web, authentication and application) and lower (SSL acceleration and firewall) level servers and intercepts all traffic (of any session using 2-factor authentication). Rather than deal with the authentication server directly, the user's traffic now goes through the Cascader, and is modified by it. This allows us to change the authentication information required from the user. The user will now have to deliver two factors of authentication to the Cascader. Note however that the Cascader does not access the legacy authentication secrets directly (although of course it can see them as they pass through), and the legacy authentication server does not have access to the new (strong) credentials. Yet, login will only succeed (along the two factor path) when the user correctly presents both factors.

B. The Trusted Device

The trusted device must be capable of basic encryption. In our implementation we used a BenQ E72 Smart-Phone running Windows Mobile version 6.0. We emphasize that we are using only the computing abilities of the device and there is no requirement that it have any connectivity at the time of login. Thus we do not require, as [6], [17], [5] do, that the device form a bluetooth connection with the untrusted machine, or that it be able to send or receive SMS, phone or web messages, as [18], [14] do. This is a major point of differentiation between our solution and previous two factor proposals based around smart phones. Furthermore, by not requiring the device to connect to the untrusted machine, we circumvent any attack where a compromised machines reads information stored in the trusted device.

1) Application Running on the Device: The application running on the device must be able to encrypt the user's legacy password using the device key K_d and the nonce N. Storing p on the device might be convenient for the user, but for the scheme to be truly two factor both p and N must be input. If the application cached p then we would really have only a single factor proof-of-possession scheme (albeit with a strong secret K_d). A screenshot of the implementation is shown in Figure 2 (note this is the version running on the simulator rather than the physical device).

2) Registering the Device: The device specific encryption key has to be transmitted to the device. The key K_d must be of high entropy, *i.e.* much higher than the highest strength allowed for the legacy password, and the desired strength of the new factor. In our sample implementation, we use a 512 bit K_d, which is downloaded to the device at registration, together with the executable code. Note that we do not assume any connectivity at the time of login: the download of the executable to the device can be via USB or any other means of porting data to the device. This is a one-time registration that never has to be repeated. At the same time the user must register his userID at the site. Only one encrypting device is permitted per user, and so the userID uniquely identifies the device key K_d that was issued.

3) Logging in to the Service: When logging in to the service the user navigates to the Cascader at, for example, www.mybank2factor.com where a login service is running. The user is first prompted for his userID, and then given a nonce N. The user must then submit $E(p, h(K_d, N))$ to the site within one minute. Upon receipt the Cascader decrypts, using the nonce it just provided and the device key it knows to have been issued to that userID:

$$E^{-1}(E(p, h(K_d, N)), h(K_d, N)) = p.$$

It forwards this to the legacy password authentication server. The overall sequence of steps is as follows and is shown in Figure 3:

1) User navigates to www.mybank2factor.com on untrusted machine
2) User enters userID
3) User is given nonce N

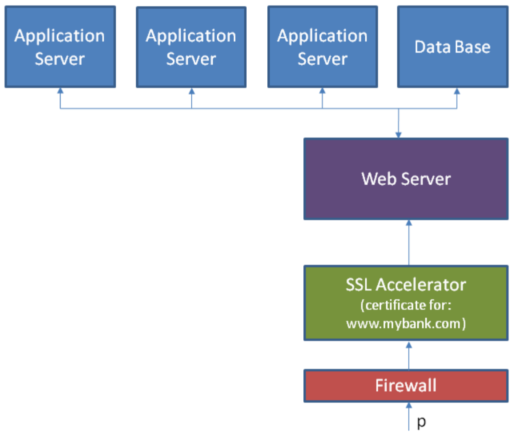


Fig. 1. Legacy system at mybank.com data center. The existing system which supports only password authentication. The web-server handles authentication and passes traffic to higher up database and application servers. Lower down is an SSL accelerator and firewall.

4) User enters N and p on trusted device
5) Trusted device calculates $E(p, h(K_d, N))$
6) User types $E(p, h(K_d, N))$ on the untrusted machine, and submits to www.mybank2factor.com

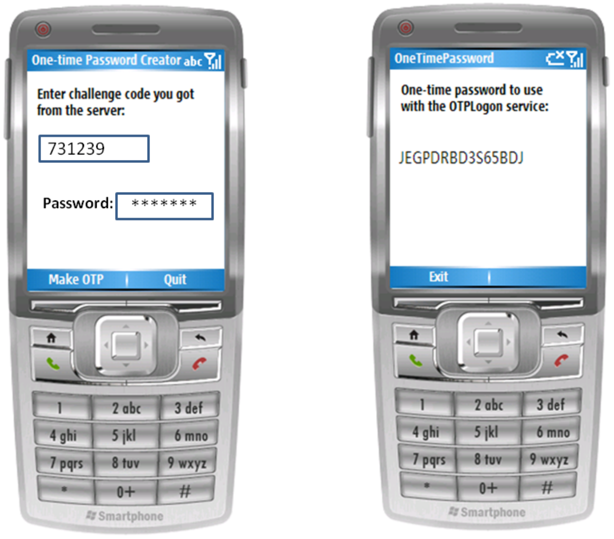
Thereafter the Cascader at www.mybank2factor.com decrypts $E^{-1}(E(p, h(K_d, N)), h(K_d, N)) = p$ and forwards, along with the userID, to www.mybank.com.

IV. Implementation and Status

A. Smart Device

1) Encryption: We use a dedicated implementation of the RC4 stream cipher to encrypt the password [23], [4]. Since RC4 does not take a separate nonce we combine the device key and nonce to produce a single-use key for each encryption. This combination is done by hashing rather than concatenation to protect against an attack involving the key schedule of RC4. Thus the actual encryption is $E(p, h(K_d, N))$, where $h()$ is the SHA-1 hash [23], [4], [21]. The nonce expires one minute after being issued. RC4 has certain known weaknesses and might easily be replaced by a more advanced stream cipher. The cipher produces a hexadecimal output. To reduce the number of characters entered we group the output 5 bits at a time and use elements from the $2^5 = 32$ character alphabet: ABCDEFGHJKLMNPQRSTUVWXYZ23456789. These characters are easily readable in most fonts on most displays without confusion (*e.g.* between '0' and 'O'). The device key K_d is stored on the device using the Data Protection API (DPAPI) [22]. The entire smart device application is approximately 1000 lines of C# code. It might easily be rewritten for other mobile devices such as iPhone *etc*.

2) Registration and Revocation of Device Keys: Key management can be as onerous a problem as changing back-end infrastructure. However, the load in our system is very light.



Enter challenge code you got from the server:

731239

Password: *******

One-time password to use with the OTPLogon service:

JEGPDRBD3S65BDJ

Fig. 2. The application was implemented on a Windows Mobile 6.0 smart-phone simulator and then ported to a BenQ E72 smart-phone. The figure depicts the simulator screenshot. The application running on the device accepts the input nonce N and produces the output code $E(p, h(K_d, N))$. Figure shows the device after user has entered N (left) and after the output code $E(p, h(K_d, N))$ has been generated (right). The user will enter the latter at the untrusted terminal.

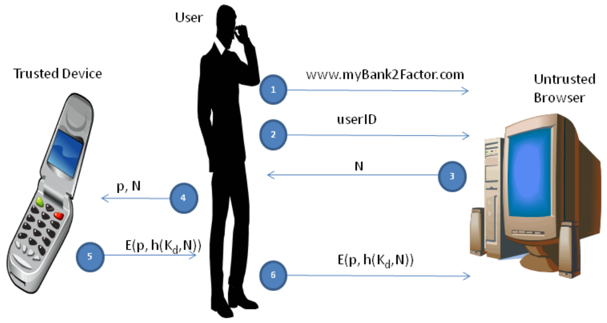


Fig. 3. The user experience illustrating the main actions taken by the user. The userID is entered on the untrusted browser, nonce N and password are entered on the smart-phone, and the resulting encryption $E(p, K_d, N)$ is entered on the browser.

Since the user already has a way of contacting the bank on a trusted channel (*i.e.* over SSL when he logs in) there is no need for a key exchange protocol. The requirement of registration is that the user's smart-phone be bound to a device key K_d. That key will be issued only once. The user downloads the application; each download of the application contains a unique K_d. This download should be available only when the user is logged in. The cascader then stores the userID-K_d pair to enable decryption later.

It is important to have a revocation procedure when the

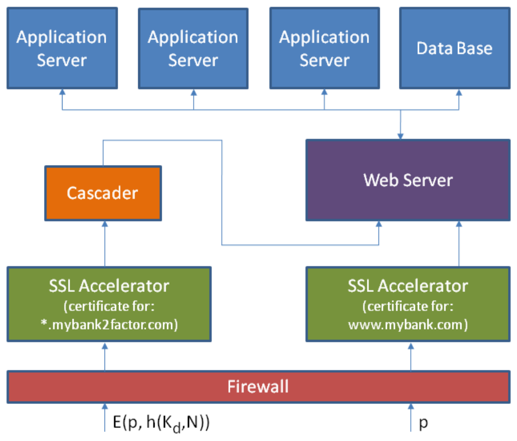


Fig. 4. The change required at the mybank.com data center. Addition of Cascader enables two factor authentication without changing any of the existing legacy components. The Cascader is added, and offers a new path for providing p to the legacy authentication system.

smart-phone is lost, stolen, broken or simply replaced. Recall that if an attacker steals the phone he cannot login without the password. A good suggestion made by Parno *et al.* [6] is that revocation should be done by phone. This simply would involve releasing the binding of the userID to the device key K_d.

Replacement is slightly more complex. Assume that a user has lost his phone, and the legacy (*i.e.* password only) path is no longer available. Thus he can no longer login (neither of course can the possessor of the phone). Currently most banks will issue a new password after authenticating the user (*e.g.* using secret questions, or asking details of the account). Since the legacy path has been severed this will no longer suffice. Instead the bank changes the password on the legacy system, and gives the user this new password, encrypted with a new key $K_d^{'}$ and a nonce. That is gives the user the code $E(p, h(K_d, N))$. Thus the user can access the account using the reset password via the cascader. At that point the user can download a new version of the application (which contains a new key). The user would then be free to register a new device by installing a new version of the device application (with a new device key K_d) on a new smart-phone.

B. Cascader

A key requirement of our approach is that the Cascader acts as a transparent proxy *without needing to reconfigure the backend servers*.

To accomplish this we leverage the work on link-translating proxies of Freedman *et al.* [16], CGIproxy [1] and Mao and Herley [19]. Mao and Herley [19] describe in detail how to build a transparent proxy which handles SSL, preserves session state, and preserves the same-origin policy separation of content. The basic idea is to map every domain to be loaded from the target server to a sub-domain of the proxy. This is done by creating a wildcard DNS entry for the Cascader, and appending the Cascader domain to the URL to be loaded. Thus if we register the domain cascade2factor.com for the Cascader then by loading https://www.paypal.com.cascade2factor.com the user is served content from https://www.paypal.com via the Cascader. In this instance the Cascader is essentially an IIS service which appends .cascader2factor.com to the URL of all absolute links in responses before relaying to the client, and strips .cascader2factor.com from all requests from the client. A detailed treatment is given in [19].

However the proxy of Mao and Herley [19] suffers from a problem with SSL content: they use a single wildcard certificate to cover multiple subdomains. Different browsers handle wildcard certificates differently (see RFC 2818): Firefox and Opera will allow a certificate for *.cascade2factor.com to cover any number of sub-domains. However, Internet Explorer, Chrome and Safari allow coverage of only a single subdomain. Thus a certificate for *.cascade2factor.com cannot cover https://www.paypal.com.cas
cade2factor.com. This has the effect that users of URRSA see many certificate errors when using IE, Chrome or Safari. Since many sites load content from several different domains (*e.g.* a user logging into hotmail will load SSL content from .hotmail.com, .live.com and .spaces.com) this generates many certificate errors for the user. This problem manifest, e.g., in URSSA [9]. We entirely eliminate this shortcoming of URRSA as follows. By restricting to a single site we have restricted the number of domains that must be covered. For example, a user logging into Paypal loads content using SSL from www.paypal.com and www.paypalobjects.com; each using a different certificate. Rather than have a single wildcard certificate to cover *.cascade2factor.com we use the subjectAltName:dNSName field of the original certificate. Thus, in the Paypal example, the first certificate would have Common Name (CN) www.paypal.com and subjectAltName:dNSName www.paypal.com.cascade2factor.com. Similarly for the paypalobjects certificate. This has several advantages over the URRSA system. First, certificate errors are entirely eliminated no matter what browser is used. Second, the user sees the same certificate whether he accesses the system via cascader or directly through the legacy path. This is an important advantage. If the site uses Extended Validation (EV) certificates that is what the user will see; if the site has a misconfigured, expired or name mismatch that fact will be displayed to the user. This contrasts with URRSA where the user sees the proxy certificate (and errors) rather than that of the web-site.

C. Status

We have implemented the entire system and successfully tested with the following banking sites: WellsFargo, Citibank, Bank of America and PayPal. Obviously we did not have access to the data centers of any of these institutions. Thus, our Cascader system is located outside the mybank.com firewall, but in other respects it follows the architecture of Figure 4 precisely. A BenQ E72 Smartphone, running Windows Mobile 6.0 was used for the trusted device. The user instals the application onto the phone, and the application already contains the (unique) device key K_d. The fact that we are able to login to sites such as Paypal attests that deployment is truly independent of the legacy servers.

After submission of the encrypted password $E(p, h(K_d, N))$ the user's browser is directed to www.paypal.com.casc
ade2factor.com and the credentials are submitted. To the user it appears exactly like a normal MyBank session; all of the functionality generally available. The two differences are that the address bar indicates www.paypal.com.cascade2factor.com instead of www.paypal.com and the certificate issued matches.

V. Attacks and Security Analysis

Loss of Codes: Most one-time password systems are vulnerable to loss of the list of OTP's. This is the case for the OTP systems [15], [12], [2], [9]. Since our codes are generated on request and are good for only one minute this is not a risk with our system. Thus, the system has similar protection to that afforded by [3].

Code Stealing: As we mentioned before, the device already has a (long) encryption key K_d. This key is specific to each account/device and will be used for every encryption done on that device. To generate an authentication string $E(p, h(K_d, N))$ the user enters the userID in the system to request an new nonce N. A natural attack is to request an extra authentication code, and store that for future use. This is the traditional code stealing attack that one-time password systems are vulnerable to. It does not succeed here: since the nonce N is useful for only one minute an attacker cannot harvest codes for future use. This is an advantage with respect to [15], [12], [2], [9]. In each of these systems there is a risk that a user is duped into revealing future codes to an attacker. By contrast securID [3] and our system survive this attack.

A slightly different version of the attack, is to choose a random N and harvest that code, hoping the system will provide the same N in a future request. The probability of success of such attack can be controlled by the length of N. In our implementation, we use a 10 digit N, and limit the number of times the system can be used to 1000. That limits the probability that a random challenge be presented in the future to 10e-7. Therefore, an attacker who successfully steals 10 codes has about 10e-6 of ever been able to use those codes. In comparison, the success probability of an attacker simply guessing the securID code is 10e-6 at each trial. We use the figure of 1000 logins in this analysis is for illustrative purposes: this would allow a user to login once a day for almost three years. That represents heavy usage, and a reasonable lifetime for a phone. Increasing the number of digits in the nonce can of course improve the security, but at the cost of forcing the user to type more.

Recall, the compromised machine has no direct access to the trusted device, except for what the user types. It cannot download or copy K_d, and it cannot request hundreds of authentication strings.

Stolen Device: An advantage of our system with respect to other two factor schemes (*e.g.* securID or smartCard + PIN) is that the proof-of-knowledge factor is never entered on the untrusted machine. An attacker who runs a keylogger on a PC and then steals the securID of a user who has logged in would then have access to both the secret and the device. By contrast an attack who steals the smart-phone from a user of our system would gain the device, but would still not know the password.

VI. Conclusion

We have shown how to convert a single factor scheme into a two-factor scheme. We create a cascade of a bank's existing legacy password system with a new server that verifies possession of a trusted device. Since the system does not require changing the legacy system it provides a painless migration path from passwords to two factor authentication.

References

[1] http://www.jmarshall.com/tools/cgiproxy.

[2] http://www.cl.cam.ac.uk/~mgk25/otpw.html.

[3] http://www.rsasecurity.com.

[4] AJ Menezes, PC Van Oorschot, SA Vanstone. Handbook of Applied Cryptography.

[5] B. Laurie and A. Singer. Choose the Red Pill and the Blue Pill. In *NSPW*, 2008.

[6] B. Parno and C. Kuo and A. Perrig. Phoolproof Phishing Prevention. In *Financial Crypto*, 2006.

[7] D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, 1999.

[8] C. Herley and D. Florêncio. How To Login From an Internet Café without Worrying about Keyloggers. *Symp. on Usable Privacy and Security*, 2006.

[9] D. Florêncio and C. Herley. One-Time Password Access to Any Server Without Changing the Server. *ISC 2008, Taipei.*

[10] D. Florêncio and C. Herley. Where Do Security Policies Come From? *SOUPS 2010, Redmond.*

[11] D. Florêncio and C. Herley. KLASSP: Entering Passwords on a Spyware Infected Machine. *ACSAC*, 2006.

[12] N. Haller. The S/KEY One-Time Password System. *Proc. ISOC Symposium on Network and Distributed System Security*, 1994.

[13] C. Herley and D. Florêncio. Nobody Sells Gold for the Price of Silver: Dishonesty, Uncertainty and the Underground Economy. *WEIS 2009, London.*

[14] R. C. Jammalamadaka, T. W. van der Horst, S. Mehrotra, K. Seamons, and N. Venkasubramanian. Delegate: A Proxy based Architecture fort Secure Website Access from an Untrusted Machine. *Proc. ACSAC*, 2006.

[15] L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 1981.

[16] M. J. Freedman and E. Freuenthal and D. Mazières. Democratizing Content Publication with Coral. *NSDI*, 2004.

[17] M. Mannan and P.C. van Oorschot. Security and Usability: The Gap in Real-World Online Banking. *NSPW*, 2007.

[18] M. Wu and S. Garfinkel and R. Miller. Secure Web Authentication with Mobile Phones. In *DIMACS Workshop on Usable Privacy and Security Software*, 2004.

[19] Z. Mao and C. Herley. A Robust Link-Translating Proxy Mirroring the Whole Web. *ACM SAC 2010.*

[20] A. Pashalidis and C. J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. *Proceedings of IEEE Globecom*, 2004.

[21] R. Anderson. Security Engineering. In *Second ed.*, 2008.

[22] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals*. Microsoft Press, fourth edition, 2005.

[23] B. Schneier. *Applied Cryptography*. Wiley, second edition, 1996.

[24] T. Moore and R. Clayton. Examining the Impact of Website Take-down on Phishing. *Proc. APWG eCrime Summit*, 2007.