The Battle Against Phishing: Dynamic Security Skins

Rachna Dhamija
University of California, Berkeley
rachna@sims.berkeley.edu

J.D. Tygar
University of California, Berkeley
tygar@cs.berkeley.edu

ABSTRACT

Phishing is a model problem for illustrating usability concerns of privacy and security because both system designers and attackers battle using user interfaces to guide (or misguide) users.

We propose a new scheme, Dynamic Security Skins, that allows a remote web server to prove its identity in a way that is easy for a human user to verify and hard for an attacker to spoof. We describe the design of an extension to the Mozilla Firefox browser that implements this scheme.

We present two novel interaction techniques to prevent spoofing. First, our browser extension provides a trusted window in the browser dedicated to username and password entry. We use a photographic image to create a trusted path between the user and this window to prevent spoofing of the window and of the text entry fields.

Second, our scheme allows the remote server to generate a unique abstract image for each user and each transaction. This image creates a "skin" that automatically customizes the browser window or the user interface elements in the content of a remote web page. Our extension allows the user's browser to independently compute the image that it expects to receive from the server. To authenticate content from the server, the user can visually verify that the images match.

We contrast our work with existing anti-phishing proposals. In contrast to other proposals, our scheme places a very low burden on the user in terms of effort, memory and time. To authenticate himself, the user has to recognize only one image and remember one low entropy password, no matter how many servers he wishes to interact with. To authenticate content from an authenticated server, the user only needs to perform one visual matching operation to compare two images. Furthermore, it places a high burden of effort on an attacker to spoof customized security indicators.

1. INTRODUCTION

Phishing is a model problem for usability concerns in privacy and security because both system designers and attackers battle in the user interface space. Careful analysis of the phishing problem promises to shed light on a wide range of security usability problems.

In this paper, we examine the case of users authenticating web sites in the context of phishing attacks. In a phishing attack, the attacker spoofs a website (e.g., a financial services website). The attacker draws a victim to the rogue website, sometimes by embedding a link in email and encouraging the user to click on the link. The rogue website usually looks exactly like a known website, sharing logos and images, but the rogue website serves only to capture the user's personal information. Many phishing attacks seek to gain credit card information, account numbers, usernames and passwords that enable the attacker to perpetrate fraud and identity theft.

Data suggest that some phishing attacks have convinced up to 5% of their recipients to provide sensitive information to spoofed websites [1]. About two million users gave information to spoofed websites resulting in direct losses of $1.2 billion for U.S. banks and card issuers in 2003 [2]. 2780 unique active phishing attack websites were reported in the month of March 2005 alone [3].

It is a dreary commentary on the state of Internet security that phishers are able to be so successful using straightforward attacks with little effort. Though we have known about spoofing vulnerabilities in browsers for years [4, 5], some browser designers initially believed that these vulnerabilities were only an academic concern that deserved little attention [5]. However, as we depend more on the Internet to conduct business and e-commerce transactions, the need to address spoofing vulnerabilities becomes more important.

The phishing problem shows that we as security designers have a distance to travel. Because both attackers and designers use user interface tools, examining this problem yields insight into usability design for other privacy and security areas.

We examine security properties that make phishing a challenging design problem in Section 2. In Section 3, we discuss a user task analysis of the skills required to detect a phishing attack. We present the design of a new authentication prototype in Section 4, analyze its security in Section 5 and discuss user testing in Section 6. We discuss related work in Section 7.

2. SECURITY PROPERTIES

Why is security design for phishing hard? As we discuss in Section 7 and elsewhere [6], a variety of researchers have proposed systems designed to thwart phishing; yet these systems appear to be of limited success. Here are some properties that come into play:

1. The limited human skills property. Humans are not general purpose computers. They are limited by their inherent skills

The authors gratefully acknowledge partial support for this work from the National Science Foundation and the United States Postal Service. The views expressed in this work are solely those of the authors and do not reflect the views of the funding sponsors.

Symposium On Usable Privacy and Security (SOUPS) 2005, July 6-8, 2005, Pittsburgh, PA, USA.

and abilities. This point appears obvious, but it implies a different approach to the design of security systems. Rather than only approaching a problem from a traditional cryptography-based security framework (e.g., "what can we secure?"), a usable design must take into account what humans do well and what they do not do well.

As an example, people often learn to screen out commonly re-occurring notices [7]. Browsers often warn users when they submit form data over an unencrypted connection. This warning is so common that most users ignore it, and some turn the warning off entirely.

2. The general purpose graphics property. Operating systems and windowing platforms that permit general purpose graphics also permit spoofing. The implications of this property are important: if we are building a system that is designed to resist spoofing we must assume that uniform graphic designs can be easily copied. As we will see in next section, phishers use this property to their advantage in crafting many types of attacks.

3. The golden arches property. Organizations invest a great deal to strengthen their brand recognition and to evoke trust in those brands by consumers. Just as the phrase "golden arches" is evocative of a particular restaurant chain, so are distinct logos used by banks, financial organizations, and other entities storing personal data. Because of the massive investment in advertising designed to strengthen this connection, we must go to extraordinary lengths to prevent people from automatically assigning trust based on logos alone.

This principle applies to the design of security indicators and icons as well. For example, users often implicitly place trust in security icons (such as the SSL closed lock icon), whether they are legitimate or not.

We revisit two properties proposed by Whitten and Tygar [8]:

4. The unmotivated user property. Security is usually a secondary goal. Most users prefer to focus on their primary tasks, and therefore designers can not expect users to be highly motivated to manage their security. For example, we can not assume that users will take the time to inspect a website certificate and learn how to interpret it in order to protect themselves from rogue websites.

5. The barn door property. Once a secret has been left unprotected, even for a short time, there is no way to guarantee that it can not been exploited by an attacker. This property encourages us to design systems that place a high priority on helping users to protect sensitive data before it leaves their control.

While each of these properties by themselves seem self-evident, when combined, they suggest a series of tests for proposed anti-phishing software. We argue that to be fully effective, anti-phishing solutions must be designed with these properties in mind.

3. TASK ANALYSIS

The Anti Phishing Working Group [APWG] maintains a "Phishing Archive" describing phishing attacks dating back to September 2003 [9]. Reviewing these reports, we constructed a task analysis of the methods and necessary skills for a user to detect a phishing attack. Space limitations prevent us from presenting the full task analysis here; it is available in a companion report [10]. Here we summarize our findings.

We find that all of the attacks exploit the human tendency to trust certain brands, logos and other trust indicators. These attacks often ironically exploit a widespread sense that the Internet is unsafe and that users must take active steps to "protect" their financial accounts and passwords. The similarity between phishing attacks, which claim that users must update passwords, account activity, etc., and legitimate security requests adds verisimilitude to phishing attacks.

The efficacy of phishing attacks is diminished when users can not reliably distinguish and verify authoritative security indicators. Unfortunately, current browser and related application programs have not been carefully designed with "security usability" in mind. As a result, users have the following problems:

Users can not reliably correctly determine sender identity in email messages. The email sender address is often forged in phishing attacks. Most users do not have the skills to distinguish forged headers from legitimate headers using today's email clients.

Users can not reliably distinguish legitimate email and website content from illegitimate content that has the same "look and feel". If images and logos are mimicked perfectly, sometimes the only cues that are available to the user are the tone of the language, misspellings or the simple fact that large amounts of personal information is being requested.

Users can not reliably parse domain names. Often they are fooled by the syntax of a domain name through "typejacking" attacks, which substitute letters that may go unnoticed (e.g. www.paypai.com and www.paypal.com), or when numerical IP addresses are used instead of text. The semantics of a domain name can also confuse users. (e.g., users can mistake www.ebay-members-security.com as belonging to www.ebay.com). Legitimate organizations heighten this confusion by using non-standard naming strategies themselves (e.g., Citibank legitimately uses *citi.com*, *citicard.com* and *accountonline.com*). Phishers have also exploited browser vulnerabilities to spoof domain names, for example by taking advantage of non-printing characters [11] and non-ascii Unicode characters [12].

Users can not reliably distinguish actual hyperlinks from images of hyperlinks. One common technique used by phishers is to display an image of a legitimate hyperlink. When clicked, the image itself serves as a hyperlink to a different rogue site. Even if the actual hyperlink is displayed in the status bar or a browser or email client, many users do not notice it.

Users can not reliably distinguish browser chrome from web page content. Browser "chrome" refers to the interface constructed by the browser around a web page (e.g., toolbars, windows, address bar, status bar). It is hard for users to distinguish an image of a window in the content of a webpage from an actual browser window. This technique has been used to spoof password dialogue windows, for example. Because the spoofed image looks exactly like a real window, a user can be fooled unless he tries to move or resize the window.

Users can not reliably distinguish actual security indicators from images of those indicators. Many users can confuse a legitimate SSL closed-lock icon, which appears on the status bar, with an image of that icon in the content of a web page. Many users simply scan for the presence of a lock icon, regardless of where it appears. Furthermore, it is hard to train users exactly where to look, because each browser uses a different icon that appears in a different location of the browser

chrome. Legitimate organizations heighten this confusion by allowing users to login from non-HTTPS pages. A form POST from a HTTP page can be delivered securely via SSL, however, there is no visual cue to indicate if the data is sent via SSL or even to the correct server (e.g., Bank of America allows users to login from its HTTP homepage. Because the page itself is not SSL protected, the bank uses an image of a lock icon near the form to indicate that it is secure).

Users do not understand the meaning of the SSL lock icon. Even if users can reliably identify a legitimate SSL lock icon on the status bar, they may be confused by what that icon actually means. The lock icon indicates that the page the user is viewing was delivered to the user securely. However, it does not guarantee that data entered into that page will also be sent securely to the server (e.g., a form on a HTTPS page may submit data to a non-HTTPS site). Some browsers provide warnings to inform the user when data is submitted insecurely, but many users ignore these warnings or turn them off.

Users do not reliably notice the absence of a security indicator. In the Firefox browser, SSL protected pages are denoted by four indicators (a closed lock icon in the status bar, text of the actual domain name in the status bar, a closed lock icon in the address bar and a yellow background in the address bar). However, in the case of non-SSL protected web pages, each of these indicators is missing. Many users do not notice the absence of an indicator, and it is trivial to insert a spoofed image of that indicator where one does not exist.

Users can not reliably distinguish multiple windows and their attributes. A common phishing technique is to place an illegitimate browser window on top of or next to a legitimate window. If they have the same look and feel, users may mistakenly believe that both windows are from the same source, regardless of variations in address or security indicators. A user may believe that the legitimate security indicators in one window also apply to the second rogue window. In the worst case, a user may not even notice that a second window exists (borderless pop-up windows make this task particularly challenging).

Users do not reliably understand SSL certificates. Very few users go through the effort of checking SSL certificates, and if they do, most do not have the skills to understand the information presented. Most users have no knowledge of certificate authorities (CAs) and what trust in a CA implies. Though users can specify the CA's that they trust to sign certificates, very few of even the most sophisticated users take this step. Some phishers have gone through the effort of registering a real certificate for their rogue phishing sites [13]. In this case, users can not rely simply on the presence of the lock icon or certificate. To detect this attack, they must be able to inspect the certificate and to distinguish the domain name of the real website from the rogue site. There are also examples where the CA certificate issuing process was subverted (e.g., Verisign issued two class 3 code-signing certificates to an individual who fraudulently claimed to be a Microsoft employee [14]).

Previous research has shown that even under normal conditions, it is difficult for average users to determine whether a browser connection is secure [15]. Our task analysis indicates that intentional spoofing attacks make this an even more challenging task for users.

4. OUR SOLUTION

4.1 Design Requirements

With the security properties and task analysis in mind, our goal is to develop an authentication scheme that does not impose undue burden on the user, in terms of effort or time. In particular, we strive to minimize user memory requirements. Our interface has the following properties:

- To authenticate himself, the user has to recognize only one image and remember one low entropy password, no matter how many servers he wishes to interact with.
- To authenticate content from a server, the user only needs to perform one visual matching operation to compare two images.
- It is hard for an attacker to spoof the indicators of a successful authentication.

We use an underlying authentication protocol to achieve the following security properties:

- At the end of an interaction, the server authenticates the user, and the user authenticates the server.
- No personally identifiable information is sent over the network.
- An attacker can not masquerade as the user or the server, even after observing any number of successful authentications.

4.2 Overview

We are developing an extension for the Mozilla Firefox browser. We chose the Mozilla platform for its openness and ease of modification. The standard Mozilla browser interface and our extension are built using Mozilla's XML-based User interface Language (XUL), a mark up language for describing user interface elements. In this section, we provide an overview of our solution before describing each component in depth.

First, our extension provides the user with a *trusted password window*. This is a dedicated window for the user to enter usernames and passwords and for the browser to display security information. We present a technique to establish a trusted path between the user and this window that requires the user to recognize a photographic image.

Next, we present a technique for a user to distinguish authenticated web pages from "insecure" or "spoofed" web pages. Our technique does not require the user to recognize a static security indicator or a secret shared with the server. Instead, the remote server generates an abstract image that is unique for each user and each transaction. This image is used to create a "skin", which customizes the appearance of the server's web page. The browser computes the image that it expects to receive from the server and displays it in the user's trusted window. To authenticate content from the server, the user can visually verify that the images match.

We implement the secure Remote Password Protocol (SRP), a verifier-based protocol developed by Tom Wu, to achieve mutual authentication of the user and the server. We chose to use SRP because it aligns well with users' preference for easy-to-memorize passwords, and it also does not require passwords to be sent over the network. We adapted the SRP protocol to allow the user and the server to independently generate the skins described above. We note that all of interface techniques we propose can be used with other underlying authentication

protocols. We also note that simply changing the underlying protocol is not enough to prevent spoofing, without also providing a mechanism for users to reliably distinguish trusted and untrusted windows.

4.3 Trusted Path to the Password Window

How can a user trust the client display when every user interface element in that display can be spoofed? We propose a solution in which the user shares a secret with the display, one that can not be known or predicted by any third party. To create a trusted path between the user and the display, the display must first prove to the user that it knows this secret.

Our approach is based on window customization [16]. If user interface elements are customized in a way that is recognizable to the user but very difficult to predict by others, attackers can not mimic those aspects that are unknown to them.

Figure 1: The trusted password window uses a background image to prevent spoofing of the window and textboxes.

Our extension provides the user with a *trusted password window* that is dedicated to password entry and display of security information. We establish a trusted path to this window by assigning each user a random photographic image that will always appear in that window. We refer to this as the user's *personal image*. The user should easily be able to recognize the personal image and should only enter his password when this image is displayed. As shown in Figure 1, the personal image serves as the background of the window. The personal image is also transparently overlaid onto the textboxes. This ensures that user focus is on the image at the point of text entry and makes it more difficult to spoof the password entry boxes (e.g., by using a pop-up window over that area).

As discussed below, the security of this scheme will depend on the number of image choices that are available. For higher security, the window is designed so that users can also choose their own personal images. Figure 1 shows examples of the trusted window with images chosen by the user.

We chose photographic images as the secret to be recognized because photographic images are more easily recognized than abstract images or text [17, 18, 19, 20, 21, 22] and because users preferred to recognize images over text in our early prototypes. However, any type of image or text could potentially be used to create a trusted path, as long as the user can recognize it. For example, a myriad of user interface elements, such as the background color, position of textboxes and font, could be randomly altered at first use to change the appearance of the window. The user can also be allowed to make further changes, however security should never rely on users being willing to customize this window themselves.

The choice of window style will also have an impact on security. In this example, the trusted window is presented as a toolbar, which can be "docked" to any location on the browser. Having a movable, rather than fixed window has advantages (because an attacker will not know where to place a spoofed window), but can also have disadvantages (because naïve users might be fooled by false windows in alternate locations). We are also experimenting with representing the trusted window as a fixed toolbar, a modal window and as a side bar.

Unlike the shared secret schemes discussed in the Related Work section, this scheme requires the user to share a secret with himself (or his browser) rather than with the server he wishes to authenticate. This scheme requires no effort on the part of the user (or a one-time customization for users who use their own images), and it only requires that the user remember one image. This is in contrast to other solutions that require users to make customizations for each server that they interact with and where the memory burden increases linearly with each additional server [16, 23, 24, 25].

4.4 Verifier Based Protocols

It is well known that users have difficulty in remembering secure passwords. Users choose passwords that are meaningful and memorable and that as a result, tend to be "low entropy" or predictable. Because human memory is faulty, many users will often use the same password for multiple purposes.

In our authentication prototype, our goal is to achieve authentication of the user and the server, without significantly altering user password behavior or increasing user memory burden. We chose to implement a verifier-based protocol. These protocols differ from conventional shared-secret authentication protocols in that they do not require two parties to share a secret password to authenticate each other. Instead, the user chooses a secret password and then applies a one-way function to that secret to generate a verifier, which is exchanged once with the other party. After the first exchange, the user and the server must only engage in a series of steps that prove to each other that they hold the verifier, without needing to reveal it.

In our prototype, we adapt an existing protocol, the Secure Remote Password protocol (SRP), developed by Tom Wu [26, 27]. SRP allows a user and server to authenticate each other over an untrusted network. We chose SRP because it is lightweight, well analyzed and has many useful properties. Namely, it allows us to preserve the familiar use of passwords, without requiring the user to send his password to the server. Furthermore, it does not require the user (or his browser) to store or manage any keys. The only secret that must be available to the browser is the user's password (which can be memorized by the user and can be low entropy). The protocol resists dictionary attacks on the verifier from both passive and active attackers, which allows users to use weak passwords safely.

Here, we present a simple overview of the protocol to give an intuition for how it works. To begin, Carol chooses a password, picks a random salt, and applies a one-way function to the password to generate a verifier. She sends this verifier and the salt to the server as a one-time operation. The server will store the verifier as Carol's "password". To login to the server, the only data that she needs to provide is her username, and the

server will look up her salt and verifier. Next, Carol's client sends a random value to the server chosen by her client. The server in turn sends Carol its own random values. Each party, using their knowledge of the verifier and the random values, can reach the same session key, a common value that is never shared. Carol sends a proof to the server that she knows the session key (this proof consists of a hash of the session key and the random values exchanged earlier). In the last step, the server sends its proof to Carol (this proof consists of a hash of the session key with Carol's proof and the random values generated earlier). At the end of this interaction, Carol is able to prove to the server that she knows the password without revealing it. Similarly, the server is able to prove that it holds the verifier without revealing it.

The protocol is simple to implement and fast. Furthermore, it does not require significant computational burden, especially on the client end. A drawback is that this scheme does require changes to the web server, and any changes required (however large or small), represent an obstacle to widespread deployment. However, there is work on integrating SRP with existing protocols (in particular, there is an IETF standards effort to integrate SRP with SSL/TLS [28]), which may make widespread deployment more feasible.

One enhancement is to only require the user to remember a single password that can be used for any server. Instead of forcing the user to remember many passwords, the browser can use a single password to generate a custom verifier for every remote server. This can be accomplished, for example, by adding the domain name (or some other information) to the password before hashing it to create the verifier [29]. This reduces memory requirements on the user, however it also increases the value of this password to attackers.

We note that simply designing a browser that can negotiate a mutual authentication protocol is not enough to stop phishing attacks, because it does not address the problem of spoofing. In particular, we must provide interaction mechanisms to protect password entry, as we addressed in section 4.3, and to help the user to distinguish content from authenticated and non-authenticated servers, as we discuss in section 4.5.

4.5 Dynamic Security Skins

Assuming that a successful authentication has taken place, how can a user distinguish authenticated web pages from those that are not "secure"? In this section we explore a number of possible solutions before presenting our own.

4.5.1 Static Security Indicators

One solution is for the browser to display all "secure" windows in a way that is distinct from windows that are not secure. Most browsers do this today by displaying a closed lock icon on the status bar or by altering the location bar (e.g., Mozilla Firefox uses a yellow background for the address bar) to indicate SSL protected sites. For example, we could display the borders of authenticated windows in one color, and insecure windows in another color. We rejected this idea because our analysis of phishing attacks suggests that almost all security indicators commonly used by browsers to indicate a "secure connection" will be spoofed. Previous research suggests that it is almost impossible to design a static indicator that can not be copied [30].

In our case, because we have established a trusted window, we could use that window to display a security indicator (such as an open or closed lock icon) or a message that indicates that the current site has been authenticated. However, this approach is also vulnerable to spoofing if the user can not easily correlate the security indicator with the appropriate window.

Figure 2: Visual hash generated by browser.

Figure 3: The trusted password window displays the visual hash that matches the website window borders.

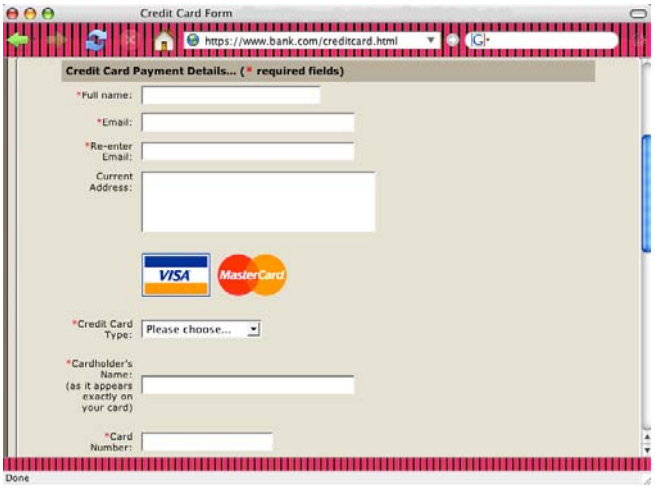

Figure 4: The browser displays the visual hash as a border around the authenticated website.

4.5.2 Customized Security Indicators

Another possibility is for the user to create a custom security indicator for each authenticated site, or one custom indicator to be used for all sites. A number of proposals require users to make per site customizations by creating custom images or text that can be recognized later [16, 23, 24, 25]. In our case, the user could personalize his trusted window, for example by choosing a border style, and the browser could display authenticated windows using this custom scheme. We rejected

this idea because it requires mandatory effort on the part of the user, and we believe that only a small number of users are willing to expend this effort. Instead, we chose to automate this process as described in the next section.

4.5.3 Automated Custom Security Indicators

We chose to automatically identify authenticated web pages and their content using randomly generated images. In this section we describe two approaches.

4.5.3.1 Browser-Generated Random Images

Ye and Smith proposed that browsers display trusted content within a *synchronized-random-dynamic boundary* [30]. In their scheme, the borders of trusted windows blink at a certain frequency in concert with a reference window.

We suggest another approach in which we randomly generate images using *visual hashes* [31]. As a visual hash algorithm, we use Random Art [32], which has previously been proposed for use in graphical password user authentication [22, 31]. Given an initial seed, Random Art generates a random mathematical formula that defines a color value for each pixel in an image. The image generation process is deterministic and the image depends only on the initial seed.

Suppose that the browser generates a random number at the start of every authentication transaction. This number is known only to the browser, and is used to generate a unique image that will only be used for that transaction. The generated image is used by the browser to create a patterned window border. Once a server is successfully authenticated, the browser presents each webpage that is generated by that server using its own unique window border. The pattern of the window border is simultaneously displayed in the user's trusted window. To authenticate a particular server window, the user only needs to ensure that two patterns match. All non-authenticated windows are displayed by the browser using a dramatically different, solid, non-patterned border, so that they can not be mistaken for authenticated windows.

There are some weaknesses in using browser window borders to distinguish "secure" pages. First, there are several ways for servers to override the display of borders. For example, it is possible for a server to open windows without any window borders. Servers can instruct the Mozilla browser to open a webpage without the browser chrome (a webpage that is not wrapped in a browser window) by issuing a simple Javascript command. Another way for servers to override the display of borders is to use "remote XUL". Remote XUL was designed to allow developers to run server based applications that do not need to be installed on the user's local machine. Normally, Mozilla uses local XUL files to build the browser interface. However, the Mozilla layout engine can also use XUL files supplied by a server to build the user interface, including content and chrome that is specified by the server.

Another disadvantage of using window borders to mark trusted content is that the border is often "far away", in terms of distance and perception, from the content of the web page that a user must trust. In some cases, it may be desirable to identify individual elements within a webpage as trusted. One possibility is for the browser to modify the display of elements within a web page (e.g., by modifying the Cascading Style Sheet file that is applied to the web page). However this approach interferes with website design and will require web designers to designate standard locations where the visual hash patterns should appear on their web pages.

We describe an approach that allows servers to mark trusted content themselves in the next section.

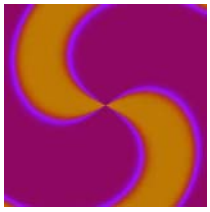

Figure 5: Visual hash generated by browser and server.

Figure 6: The trusted password window displays the visual hash that matches the website background.

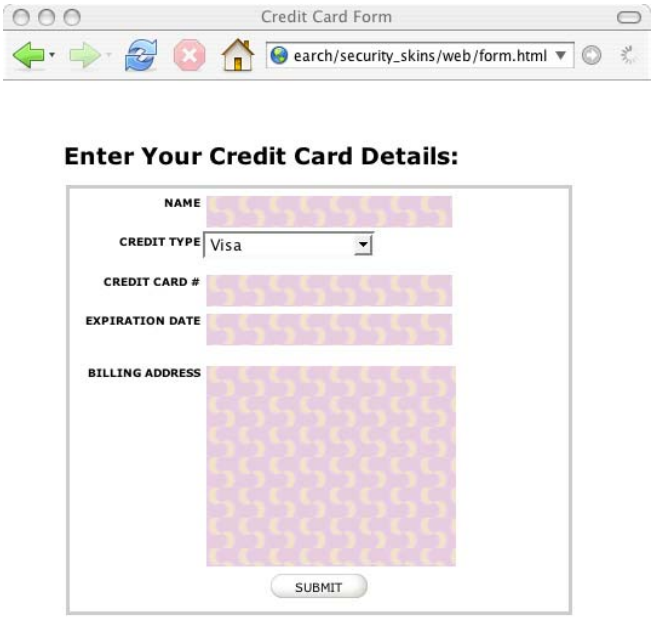

Figure 8: The website displays the visual hash as a background of a form element.

4.5.3.2 Server-Generated Random Images

In this section, we describe an approach for the *server* to generate images that can be used to mark trusted content.

To accomplish this, we take advantage of some properties of the SRP protocol (use of this specific protocol is not a requirement for our approach). In the last step of the protocol, the server presents a hash value to the user, which proves that the server holds the user's verifier. In our scheme, the server uses this value to generate an abstract image, using the visual hash algorithm described above. The user's browser can independently reach the same value as the server and can compute the same image (because it also knows the values of the verifier and the random values supplied by each party). The browser presents the user with the image that it expects to receive from the server in the trusted password window. Neither the user nor the server has to store any images in advance, since images are computed quickly from the seed.

The server can use the generated image to modify the content of its webpage in many ways. The remote server can create a border around the entire web page or can embed the image within particular elements of the webpage. For example, when requesting sensitive personal information from a user, a website can embed the image in the background of a form, as shown in Figure 8. This provides the user with a means to verify that the information request originates from a known party.

Websites must be carefully designed to use images in a way that does not clutter the design or create confusion for the user. User testing is required to determine the actual entropy of the image generation process, that is, how distinguishable patterns are between the images that are generated.

4.6 User Interaction

In this section we describe the process of a user logging in to his bank website.

The first time the browser is launched, it displays the user's trusted password window with a randomly chosen photographic image. The user can choose to keep the assigned image or can select another image.

During a set-up phase, the user chooses an easy to memorize password (it may be low entropy). The browser computes a one-way function on this password to generate a verifier, which is sent to the bank as a one-time operation. The verifier can be sent to the bank online, in the same manner that user passwords are supplied today, or through an out of band transaction, depending on security requirements. If the verifier is sent online, the process must be carefully designed so that the user can not be tricked into providing it to a rogue site.

At each login, the user must recognize his personal image and enter his username and password into the trusted window. The password is used to generate the verifier, however neither the password nor the verifier are sent to the bank. The only personal data that the bank requires at each login is the username. In the background, the client then negotiates the SRP protocol. If authentication is successful, the trusted window will display the image that it expects to receive from the bank.

Importantly, our browser extension also sets some simple browser window display preferences to prevent and detect spoofed windows. For example, the browser does not allow any windows to be placed on top of the trusted password window. Additionally, all windows not generated by the authenticated server can have a dramatically different appearance that users can specify (e.g., they will be greyed out).

The advantage from the user's point of view is that only one check (a visual match of two images) is required to establish the identity of the server (or more specifically, to establish that this is an entity that she has communicated with before). The disadvantage to the user is that the action of matching two images is more cumbersome than quickly scanning for the presence of a static binary "yes/no" security indicator. However, we expect image matching to be less cumbersome and more intuitive to users than inspecting a certificate, for example. We will perform user testing as described below to discover how cumbersome this interaction technique is for users, if users are able to perform verification through image matching and if users can detect spoofed windows.

5. SECURITY ANALYSIS

In this section, we discuss the vulnerability of our scheme to various attacks.

5.1 Leak of the Verifier

The user's verifier is sent to the bank in a one-time operation. Thereafter, the user must only supply his password to the browser and his username to the server to login.

The server stores the verifier, which is based on the user's password but which is not *password-equivalent* (it can not be used as a password). Servers are still required to guard the verifier to prevent a dictionary attack. However, unlike passwords, if this verifier is stolen (by breaking into the server database or by intercepting it the one time it is sent to the bank), the attacker does not have sufficient information to impersonate the user, which makes the verifier a less valuable target to phishers.

If a verifier is captured it can, however, be used by an attacker to impersonate the bank to one particular user. Therefore, if the verifier is sent online, the process must be carefully designed so that the user can not be tricked into providing it to a rogue site.

5.2 Leak of the Images

Our scheme requires two types of images, the *personal image* (a photographic image assigned or chosen by the user) and the *generated image* used to create the security skin. The user's personal image is never sent over the network and only displayed to the user. Therefore, the attacker must be physically present (or must compromise the browser) to observe or capture the personal image.

If the generated image is observed or captured, it can not be replayed in subsequent transactions. Furthermore, it would take an exhaustive dictionary attack to determine the value that was used to generate the image, which itself could not be used to not reveal anything about the password.

5.3 Man-in-the-Middle Attacks

SRP prevents a classic man-in-the middle attack, however a "visual man-in-the-middle" attack is still possible if an attacker can carefully overlay rogue windows on top of the trusted window or authenticated browser windows. As discussed in Section 4, we have specifically designed our windows to make this type of attack very difficult to execute.

5.4 Spoofing the Trusted Window

Because the user enters his password in the trusted password window, it is crucial that the user be able to recognize his own

customized window and to detect spoofs. If the number of options for personalization is limited, phishers can try to mimic any of the available choices, and a subset of the population will recognize the spoofed setting as their own (especially if there is a default option that is selected by many users). If an attacker has some knowledge of the user, and if the selection of images is limited, the choice of image may be predictable [33]. In addition to a large number of randomly assigned personal images, we will encourage unique personalization (e.g., allow the users to use their own photos). User testing is needed to determine if users can be trained to only enter their passwords when their own personal image shown.

5.5 Spoofing the Visual Hashes

If this system were widely adopted, we expect that phishers will place false visual hashes on their webpages or webforms to make them appear secure. Users who do not check their trusted window, or users who fail to recognize that their personal image is absent in a spoofed trusted window, could be tricked by such an attack. It is our hope that by simplifying the process of website verification, that more users (especially unsophisticated users) will be able to perform this important step.

5.6 Public Terminals and Malware

A user can login from any location, with the browser extension installed, by supplying his password. However, a user can not ensure that the password window can be trusted without also saving his personal image in the browser. In future work, we will investigate how to protect users in locations where they are not able to store the personal image (e.g., public terminals).

This scheme may provide some protection against pharming attacks, where cache poisoning is used to redirect users to rogue websites. If the user only enters his password into the trusted window, rogue websites will not be able to capture that information. However, this scheme does not address phishing threats that arise from malware installed on the users machine (e.g., keylogging software). To prevent malware attacks, an area for future work is to develop trusted paths between the user and the operating system.

6. USER TESTING

A successful interface should allow the user to easily login to a website, to verify that the website is indeed the one it claims to be and to detect spoof attempts. In this section, we describe our design process, informal testing and our plans for formal user testing.

6.1 Iterative Design and Informal Testing

We are using an iterative design process, where users are invited to informally interact with mock-ups of our interface at several stages during the design process.

Soliciting user feedback has resulted in many changes to the interface. For example, Figure 9 displays an earlier mockup of our prototype. In this case, user selected text was used as the recognized element to create a trusted path to the password window. We rejected this idea because users found it easier to recognize images than text. Early prototypes also allowed users to choose the random number that is provided by the browser to the server during SRP authentication, but users were confused by this concept.

Overall, users responded favorably to the use of a customized window that is dedicated to password entry. In particular, users find the use of photographic images in the trusted password window appealing. Many users indicated a preference for choosing their own images. Making it very easy for users to add new images will improve security (by increasing the workload for an attacker to spoof those images). However, it may also weaken security if users can be tricked in choosing a particular image.

Users do seem to grasp the concept of a website "proving" its identity by displaying an image, and users have been able to successfully match images that are presented on the website to those that appear in their trusted window.

We experimented with many trusted window designs to test for user acceptance and resistance to spoofing. The security of the trusted window is linked to recognition of the personal image. To protect the generated image from being spoofed, it must also be linked to the personal image. Users did not like trusted window designs where the generated image was embedded in the personal image or where the personal image was obscured. We settled on a design where the generated image is both transparently visible underneath the personal image and fully visible as a border or frame around the personal image. Further design work is needed to optimize the size of the images to support the recognition and matching tasks, while minimizing the size of the window. We also need to determine the best format and placement for the window that supports current browsing behavior, interaction with trusted websites and that minimizes spoofing.

We adjusted the transparency of the images over the textboxes so that they maximized visibility of the image while minimizing interference with text entry. Some users expressed a desire to customize the placement of the textboxes, so that certain portions of the image are more visible.

We will continue to develop the prototype and to experiment with interaction mechanisms (e.g., how to communicate authentication success and failures, how to support multiple DSS sites, how to support interaction with both trusted and non-trusted sites, how to display images on websites and practical details such as creating accounts, changing passwords and how to interact with the existing password manager). We also have a number of important questions to address about user behavior: Can users be trained to only enter their passwords in a separate window and only when their own personal image shown? Can users reliably verify websites through image matching? Can users reliably detect spoofed windows and websites? Our formal study will investigate whether users can perform these tasks in practice.

6.2 Formal Experimental User Study Design

Much of the evidence we have on the success of spoofing and anti-spoofing techniques is anecdotal, and there is very little empirical data on this subject.

We will conduct an experimental user study to evaluate the effectiveness of image comparison and display customization as techniques for users to identify remote servers. We are currently designing a between-subjects study to compare our prototype to other techniques. In the study, participants will be asked to create an account on a remote server and to login. We will periodically send the users email that asks them to login to the website in order manage their funds (which is their payment for participation in the study). Occasionally, users

will be sent to a website that spoofs the content of the site as well as the security indicators (such as their trusted window). Participants will be divided into three groups, one using Dynamic Security Skins, one using a shared secret scheme and another using only a standard SSL equipped browser (the control condition). Effectiveness of the prototype will be measured by the performance and error rate in account creation and login tasks, the ability for users to authenticate legitimate servers, the rate of detecting spoof attempts and user satisfaction.

Additionally, we plan to release the application to the public for widespread testing.

Figure 9: Early mockup displays a visual hash that can be matched to the remote website

7. RELATED WORK

The rapid growth in phishing attacks has spurred calls for solutions. A number have been proposed ranging from quick-fix changes to more substantial redesigns. In this section, we provide an overview of the anti-phishing proposals. We illustrate where the proposals ignore or address the security properties developed in Section 2 (the limited human skills property, general purpose graphics property, the golden arches property, the unmotivated user property and the barn door property).

In general, attempts to solve the phishing problem can be divided into three approaches: third party certification and direct authentication, and phishing specific tools.

7.1 Third Party Certification

7.1.1 Hierarchical and Distributed Trust Models

Third party certification includes hierarchical trust models, like Public Key Infrastructure (PKI), which has long been proposed as a solution for users to authenticate servers and vice-versa. In PKI, chains of Certificate Authorities (CAs) vouch for identity by binding a public key to a entity in a digital certificate. The Secure Sockets Layer (SSL) and Transport Layer Security (TLS), its successor, both rely on PKI.

In the typical use of SSL today only the server is authenticated. SSL also supports mutual authentication, and in theory it is possible for both servers and users to obtain certificates that are signed by a trusted CA. Though it is an active area of research, there is currently no practical scheme for widely deploying signed personal certificates. A further challenge is how to handle the revocation of credentials. The widespread use of personal certificates may also raise privacy concerns due to the personally identifiable information contained in certificates. Even with the wide use of one-sided SSL that is in place today (in the form of server digital certificates signed by a trusted CA), there are problems. As we examined in our task analysis, certificates have been falsely issued, and most users do not have the knowledge or skill to understand digital certificates and the delegation of trust. Therefore, SSL as it is implemented in browsers today, ignores all of the properties discussed in Section 2.

Other third party approaches include "web of trust" distributed trust models (e.g., Pretty Good Privacy [34]) and the use of third party seals to indicate trusted websites (e.g. Verisign Seal Program [35] and TRUSTe [36]). By displaying seals as graphics that can be easily copied, trusted seal programs ignore the "general purpose graphics" property.

7.1.2 Trustbar

The "Trustbar" proposal is a third party certification solution, where websites logos are certified. The authors suggest creating a "trusted credentials area" as a fixed part of the browser window [37]. This area can be used to present credentials from the website, such as logos, icons and seals of the brand, that have been certified by trusted certificate authorities or by peers using a PGP "web of trust". A strength of the solution is that it does not rely on complex security indicators. However, we must consider the "general purpose graphics" and "golden arches" properties. Because the logos do not change, they can be easily copied and the credentials area of the browser can be spoofed (e.g., an attacker can draw an image of the credentials area into the top portion of an untrusted webpage to make it appear trusted). Therefore, careful consideration must be given to the design of an indicator for insecure windows so that spoofed credentials can be easily detected. It is not clear how logos will be certified and how disputes will be resolved in the case of similar logos.

7.2 Direct Authentication

Direct authentication approaches include user authentication and server authentication schemes.

7.2.1 Multi-Factor User Authentication

These schemes use a combination of factors to authenticate the user. The factors can be something you know (e.g., a password or PIN), something you have (e.g., a token or key) or something you are (e.g., biometrics).

7.2.1.1 AOL Passcode

America Online's Passcode has been proposed as a phishing defense. This program distributes RSA SecurID devices to AOL members [38, 39] . The device generates and displays a unique six-digit numeric code every 60 seconds, which can be used as a secondary password during login to the AOL website. This scheme reduces the value of collecting passwords for attackers because the passwords can not be used for another transaction. It does not, however, prevent a man-in-the-middle (MITM) attack where the attacker lures a user to a spoofed AOL website to collect both the primary and secondary passwords. These passwords can immediately be presented by the attacker to AOL in order to masquerade as the user. The Passcode program does raise the bar for phishing attacks today, but we expect that if the bar is raised everywhere, phishers will soon turn to this type of "live" MITM attack. By not providing the user with any

means to verify the correct identity of the server, this scheme ignores the "limited human skills" property.

7.2.1.2 Secondary SMS Passwords

Other two factor user-authentication schemes, such as issuing secondary passwords to users via Short Message Service (SMS) text messages on their cell phones [40] are also vulnerable to MITM attacks. In general, two factor user authentication schemes serve to protect the server from fraud, rather than protecting the user from phishing attacks if they do not provide a mechanism for the user to authenticate the server. This ignores the "limited human skills" property.

7.2.2 Server Authentication Using Shared Secrets

7.2.2.1 Passmark and Verified by Visa

Shared-secret schemes have been proposed as one simple approach to help users identify known servers. For example in proposals such as Passmark [23] and Verified by Visa [25], the user provides the server with a shared secret, such as and image and/or passphrase, in addition to his regular password. The server presents the user with this shared secret, and the user is asked to recognize it before providing the server with his password.

The most obvious weakness of shared secret schemes is that the server must display the shared secret in order to authenticate itself to the user. If the secret is observed or captured, the image can be replayed until the user notices and changes it.

In the Passmark scheme, the bank server places a secure cookie on user machine, which must be presented at login. This prevents a classic man-in-the middle (MITM) attack where an attacker interposes himself between the client and the bank. However, it does not prevent the attack in which a rogue server instructs the browser to display two windows. The first window displays the real login page of the legitimate bank, which has legitimate trust indicators, such as the lock icon or the shared secret (e.g., the passmark). A second window is also opened that displays a webpage from the rogue server. By careful placement of the window, an attacker can convince the user to supply his password.

There is a much easier way to trick the user into revealing his password and passmark. This involves spoofing the passmark "re-registration" process. If the user wishes to login using a new computer, or if the secure cookie has been deleted, the user must "re-register" his passmark. In this case, the user is shown a "passmark not shown" screen and must *enter the password* in order to register. Therefore, an attacker can direct users to a screen that *claims* that the cookie has been deleted or does not exist. The legitimate error page asks users to ensure that they have reached this page by typing in the URL by hand, however a spoofed error page will probably not include this warning. A number of attacks are possible that require more difficulty (e.g., breaking the secure cookie, physical observation of the secret image, discovering the potential range of images and then guessing the image). However, spoofing is likely to require the least amount of effort to defeat the most people, and we expect that this type of spoofing attack will become common if systems like Passmark are widely deployed.

There is evidence that suggests that users are able to correctly recognize a large number of images [18]. However, if a user is required to remember different images or passphrases for a number of different servers, any difficulty in recognizing an image can be exploited by an attacker. This scheme ignores the "limited human skills", "general purpose graphics" and "golden arches" properties.

7.2.3 Server Authentication Using Self-Shared Secrets

In this section, we examine web server authentication schemes that require the user to share a secret with his own device (e.g., web browser) rather than with the web server.

7.2.3.1 SRD

Ye and Smith proposed "Synchronized Random Dynamic Boundaries" to secure the path from users to their browser. [30]. This scheme uses a random number generator to set a bit that determines whether the browser border is inset or outset. The browser border alternates between inset and outset at a certain frequency in concert with a reference window. A strength of this solution is that it recognizes the "general purpose graphics" problem. In this scheme, rogue servers can not predict the random number that is chosen by the browser, and therefore it is difficult to create spoof windows that blink at the correct frequency. A weakness of this approach is that it ignores the "limited human skills" property; dynamically "blinking" borders may not be easily distinguished by users, and frequent border changes are likely to be distracting. The security depends on how many border frequency options are available and how many users can differentiate.

7.2.3.2 YURL Petnames

In the YURL proposal, the user's browser maintains a mapping of a public key hash to petname. When a user visits a page identified by a YURL, the browser displays the petname that the user previously associated with the public key hash [24]. An untrusted site can be recognized by the absence of a corresponding petname displayed in the browser. This is a very simple scheme that requires a small degree of personalization for each website. This scheme ignores the "unmotivated user property" because security relies on users to be motivated to customize petnames for trusted sites.

One advantage of this scheme is that the secret (the petname) is shared with the user's browser (rather than with the trusted server). Careful consideration must be given to the design of the untrusted state. That is, untrusted windows should be clearly marked as having no petname. Otherwise, attackers can spoof the petname display area in the browser and fool many users.

The "limited human skills" property is also important. Because pet names rely on user memory to recognize the secret phrase and to associate it with the correct website, we expect that users will choose predictable petnames. For example, many users will choose "Amazon" for Amazon.com. The designers can encourage users to select unique petnames to improve spoof-resistance.

7.3 Anti-Phishing Tools

7.3.1 eBay Toolbar

The eBay Toolbar is a browser plug-in that eBay offers to its customers to help keep track of auction sites [41]. The toolbar has a feature, called AccountGuard, which monitors web pages that users visit and provides a warning in the form of a colored tab on the toolbar. The tab is usually grey, but turns green if the user is on an eBay or PayPal site or red if the user is on a site that is known to be a spoof by eBay. The toolbar also allows users to submit suspected spoof sites to eBay. One

drawback to this approach is that it only applies to eBay and PayPal websites. Users are unlikely to want to use several types of toolbars (though it may be possible to develop a toolbar that would work for a range of sites). The main weakness is that there will always be a period of time between the time a spoof is detected and when the toolbar can begin detecting spoofs for users. If spoofs are not carefully confirmed, denial of service attacks are possible. This implies that some percentage of users will be vulnerable to spoofing. For these users, "the barn door" property implies that their personal data will not be protected.

7.3.2 SpoofGuard

SpoofGuard is an Internet Explorer browser plug-in that examines web pages and warns users when a certain page has a high probability of being a spoof [42]. This calculation is performed by examining the domain name, images and links and comparing them to the stored history and by detecting common characteristics of spoofed websites. If adopted it will force phishers to work harder to create spoof pages. However, SpoofGuard needs to stay one step ahead of phishers, who can test their webpages against SpoofGuard. New detection tests will continuously need to be deployed as phishers become more sophisticated.

SpoofGuard makes use of PwdHash [29], an Internet Explorer plug-in that replaces a users password with a one way hash of the password and the domain name. As a result, the web server only receives a domain-specific hash of the password instead of the password itself. This is a simple but useful technique in addressing the "barn door property" and preventing phishers from collecting user passwords. Both SpoofGuard and PwdHash ignore the "general purpose graphics" property by using a security indicator (a traffic light) that can be easily copied.

7.3.3 Spoofstick

Spoofstick is a toolbar extension for Internet Explorer and Mozilla Firefox that provides basic information about the domain name of the website. For example, if the user is visiting Ebay, the toolbar will display "You're on ebay.com". If the user is at a spoofed site, the toolbar might instead display "You're on 10.19.32.4". This toolbar can help users to detect attacks where the rogue website has a domain name that syntactically or semantically similar to a legitimate site. Unfortunately, the current implementation of Spoofstick can be fooled by clever use of frames when different websites are opened in multiple frames in the browser window [43]. This ignores the "limited human skills" property, because users must be aware of the use of hidden frames on a webpage. Spoofstick does address the "general purpose graphics" property by allowing users to customize the appearance of the toolbar.

8. REFERENCES

[1] Loftesness, Scott, *Responding to "Phishing" Attacks*. 2004, Glenbrook Partners, http://www.glenbrook.com/opinions/phishing.htm

[2] Litan, Avivah, *Phishing Attack Victims Likely Targets for Identity Theft*, in *Gartner First Take FT-22-8873*. 2004, Gartner Research

[3] Anti-Phishing Working Group, *Phishing Activity Trends Report March 2005*, http://antiphishing.org/APWG_Phishing_Activity_Report_March_2005.pdf

[4] Ed Felten, D. Balfanz, D. Dean, D. Wallach, *Web Spoofing: An Internet Con Game*. Proceedings of the 20th Information Security Conference, 1996.

[5] Bugzilla, *Bugzilla Bug 22183 - UI spoofing can cause user to mistake content for chrome (bug reported 12/20/1999, publicly reported 7/21/2004)*, https://bugzilla.mozilla.org/show_bug.cgi?id=22183

[6] Rachna Dhamija, J.D. Tygar, *Phish and HIPs: Human Interactive Proofs to Detect Phishing Attacks*. Proceedings of the 2nd International Workshop on Human Interactive Proofs (HIP05), Springer Verlag Lecture Notes in Computer Science, 2005.

[7] Nathan Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, Joseph Konstan, *Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware*. Proceedings of the Symposium on Usable Privacy and Security, 2005.

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

[9] Anti-Phishing Working Group, APWG Phishing Archive, http://anti-phishing.org/phishing_archive.htm

[10] Dhamija, Rachna, *Detecting Phishing Attacks: A User Task Analysis*. Authentication for Humans: Designing and Evaluating Usable Security Systems. forthcoming.

[11] Secunia, *Internet Explorer URL Spoofing Vulnerability*. 2004, http://www.microsoft.com%00@secunia.com/advisories/10395/

[12] Secunia, *Multiple Browsers Vulnerable to the IDN Spoofing Vulnerability*. 2005, http://secunia.com/multiple_browsers_idn_spoofing_test

[13] Netcraft, *SSL's Credibility as Phishing Defense is Tested*. 2004, http://news.netcraft.com/archives/2004/03/08/ssls_credibility_as_phishing_defense_is_tested.html

[14] Microsoft, *Microsoft Security Bulletin MS01-017*, in *Erroneous Verisign-Issued Digital Certificates Pose Spoofing Hazard*. 2001, http://www.microsoft.com/technet/security/bulletin/MS01-017.mspx

[15] Batya Friedman, David Hurley, Daniel Howe, Edward Felten, Helen Nissenbaum, *Users' Conceptions of Web Security: A Comparative Study*. CHI 2002 Extended Abstracts of the Conference on Human Factors in Computing Systems, 2002: p. 746-747.

[16] J.D. Tygar, Alma Whitten, *WWW Electronic Commerce and Java Trojan Horses*. Proceedings of the Second USENIX Workshop on Electronic Commerce, 1996.

[17] A. Paivio, K. Csapo, *Concrete Images and Verbal Memory Codes*. Journal of Experimental Psychology, 1969. **80**(2): p. 279-285.

[18] Haber, Ralph Norman, *How we remember what we see*. Scientific American, 1970. **222**(5): p. 104-112.

[19] Intraub, Helene, *Presentation rate and the representation of briefly glimpsed pictures in memory*. Journal of Experimental Psychology: Human Learning and Memory, 1980. **6**(1): p. 1-12.

[20] L. Standing, J. Conezio, R. Haber, *Perception and Memory for Pictures: Single-trial learning of 2500 visual stimuli.* Psychonomic Science, 1970. **19**: p. 73-74.

[21] Shepard, R., *Recognition Memory for Words, Sentences and Pictures.* Journal of Verbal Learning and Verbal Behavior, 1967. **6**(156-163).

[22] Rachna Dhamija, Adrian Perrig, *Deja Vu: A User Study Using Images for Authentication.* Proceedings of the 9th USENIX Security Symposium, 2000.

[23] PassMark Security, *Protecting Your Customers from Phishing Attacks- An Introduction to PassMarks*, http://www.passmarksecurity.com/

[24] Waterken Inc., *Waterken YURL Trust Management for Humans*, http://www.waterken.com/dev/YURL/Name/

[25] Visa, Verified by Visa, http://www.visa.com/

[26] Wu, T., *The Secure Remote Password Protocol.* Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, San Diego, CA, 1998: p. 97-111.

[27] Wu, T., *SRP-6: Improvements and Refinements to the Secure Remote Password Protocol*. 2002: Submission to the IEEE P1363 Working Group

[28] D. Taylor, T. Wu, N. Mavroyanopoulos, T. Perrin, *Using SRP for TLS Authentication draft-ietf-tls-srp-08*. 2004, IETF TLS Working Group: http://www.ietf.org/internet-drafts/draft-ietf-tls-srp-08.txt

[29] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, John C. Mitchell, *A Browser Plug-in Solution to the Unique Password Problem.* Proceedings of the 14th Usenix Security Symposium, 2005.

[30] Zishuang Ye, Sean Smith, *Trusted Paths for Browsers.* Proceedings of the 11th Usenix Security Symposium, 2002.

[31] Adrian Perrig, Dawn Song, *Hash Visualization: A New Technique to Improve Real World Security.* Proceedings of the International Workshop on Cryptographic Techniques and E-commerce, 1999.

[32] Bauer, Anrej, *Random Art*, http://gs2.sp.cs.cmu.edu/art/random/

[33] Darren Davis, Fabian Monrose, Michael Reiter, *On User Choice in Graphical Password Schemes.* Proceeding of the 13th Usenix Security Symposium, 2004.

[34] Pretty Good Privacy (PGP), http://www.pgp.com/

[35] Verisign, *Verisign Secured Seal Program*, http://www.verisign.com/products-services/security-services/secured-seal/

[36] TrustE, http://www.truste.org/

[37] Amir Herzberg, Ahmad Gbara, *TrustBar: Protecting (even Naive) Web Users from Spoofing and Phishing Attacks*. 2004: Cryptology ePrint Archive: Report 2004/155

[38] RSA Security, *America Online and RSA Security Launch AOL PassCode Premium Service*. 2004, http://www.rsasecurity.com/press_release.asp?doc_id=5033

[39] RSA Security, *Protecting Against Phishing by Implementing Strong Two-Factor Authentication*. 2004, https://www.rsasecurity.com/products/securid/whitepapers/PHISH_WP_0904.pdf

[40] Pullar-Strecker, Tom, *NZ bank adds security online*, in *The Sydney Morning Herald*. November 8, 2004: Wellington.

[41] eBay, *eBay Toolbar*, http://pages.ebay.com/ebay_toolbar/

[42] Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, John C. Mitchell, *Client Side Defense Against Web-based Identity Theft*, http://crypto.stanford.edu/SpoofGuard/#publications

[43] Core Street, *Spoofstick*, http://www.corestreet.com/spoofstick/