In Phishing and Countermeasures, eds. M. Jakobsson and S. Myers. Wiley-Interscience, 2007, pp. 339-351

9.5 DYNAMIC SECURITY SKINS

Rachna Dhamija and J. D. Tygar

Phishing is a model problem for illustrating usability concerns of privacy and security because both system designers and attackers battle with user interfaces to guide (or misguide) users. Careful analysis of the phishing problem promises to shed light on a wide range of security usability problems. In this chapter, 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 begin by examining security properties that make phishing a challenging design problem in 9.5.1. In 9.5.2, we summarize the results of a usability study evaluating why phishing attacks work. We present the design of a new authentication prototype in 9.5.3, discuss the user interaction in 9.5.4 and present a security analysis in 9.5.5.

In Phishing and Countermeasures, eds. M. Jakobsson and S. Myers. Wiley-Interscience, 2007, pp. 339-351

9.5.1 Security Properties

Why is security design for phishing hard? Building on the work of Whitten and Tygar [51], we identify eight properties of computer security that make usability difficult:

1. **The limited human skills property.** Humans are not general purpose computers. They are limited by their inherent skills 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 reoccurring notices [21]. 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. 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.

4. **The unmotivated user property.** Security is usually a secondary goal. Most users prefer to focus on their primary tasks, and therefore designers cannot expect users to be highly motivated to manage their security. For example, we cannot 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 cannot 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.

9.5.2 Why Phishing Works

The Anti-Phishing Working Group maintains a "Phishing Archive" describing phishing attacks dating back to September 2003 [3]. We performed a cognitive walkthrough on the approximately 200 sample attacks within this archive to develop a set of hypotheses about how users are deceived. We tested these hypotheses in a usability study: We showed 22 participants 20 websites and asked them to determine which ones were fraudulent, and why. Details are available in [17, 18]. Our key findings are:

- Good phishing websites fooled 90% of participants.
- Existing antiphishing browsing cues are ineffective: 23% of participants in our study did not look at the address bar, status bar, or any SSL indicators.
- On average, our participant group made mistakes on our test set 40% of the time.
- Popup warnings about fraudulent certificates were singularly ineffective: 15 out of 22 participants proceeded without hesitation when presented with these warnings.
- The indicators of trust presented by the browser are trivial to spoof. By using very simple spoofing attacks, such as copying images of browser chrome or the SSL indicators in the address bar or status bar, we were able to fool even our most careful and knowledgeable users.
- Participants proved vulnerable across the board to phishing attacks. In our study, neither education, age, sex, previous experience, nor hours of computer use showed a statistically significant correlation with vulnerability to phishing.

Our study suggests that a different approach is needed in the design of security systems. In the next section, we propose a new approach, that allows a remote web server to prove its identity in a way that is easy for a human user to verify (exploiting the ability of users to recognize and match images), but hard for an attacker to spoof.

9.5.3 Dynamic Security Skins

9.5.3.1 Design Requirements With the security properties and usability study in mind, our goal was 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 cannot masquerade as the user or the server, even after observing any number of successful authentications.

In Phishing and Countermeasures, eds. M. Jakobsson and S. Myers. Wiley-Interscience, 2007, pp. 339-351

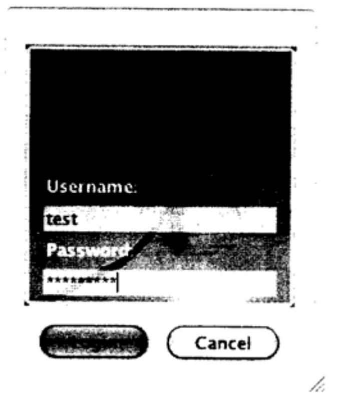


Figure 9.15 The trusted password window uses a background image to prevent spoofing of the window and textboxes.

9.5.3.2 *Overview* We developed a prototype of our scheme as 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 "unsecure" 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 made use of the use of the Secure Remote Password protocol (SRP) [55], to achieve mutual authentication of the user and the server. We propose an adaptation of 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.

9.5.3.3 *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.

We made use of an existing protocol, the Secure Remote Password protocol (SRP), developed by Tom Wu [55]. 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. Her client 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's client 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's client 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's client (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), 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 [42]. 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 and to help the user to distinguish content from authenticated and non-authenticated servers.

9.5.3.4 *Trusted Path to the Password Window* In order to authenticate, Carol must correctly supply her password to her client (the browser) and not to a rogue third party. 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 cannot 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 [48]. If user interface elements are customized in a way that is recognizable to the user but very difficult to predict by others, attackers cannot mimic those aspects that are unknown to them.

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 9.15, 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.

We chose photographic images as the secret to be recognized because photographic images are more easily recognized than abstract images or text [44, 22, 45, 23, 15, 16] 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 naive users might be fooled by false windows in alternate locations). We are currently experimenting with representing the trusted window as a fixed toolbar, a modal window, and a side bar.

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 recognize 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 [48, 39, 49, 50].

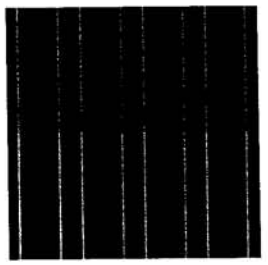


Figure 9.16 An example of a visual hash that is generated by browser.

9.5.3.5 *Distinguishing Secure Web Pages* 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.

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 cannot be copied [58].

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 cannot easily correlate the security indicator with the appropriate window.

User 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 [48, 39, 49, 50]. 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.

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.

Browser-Generated Random Images. Ye and Smith proposed that browsers display trusted content within a synchronized-random-dynamic boundary [58]. In their scheme, the borders of trusted windows blink at a certain frequency in concert with a reference window.

In Phishing and Countermeasures, eds. M. Jakobsson and S. Myers. Wiley-Interscience, 2007, pp. 339-351

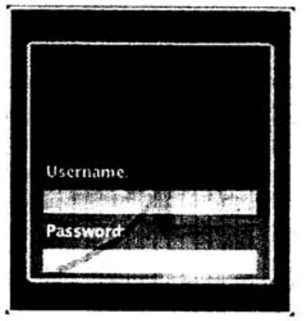
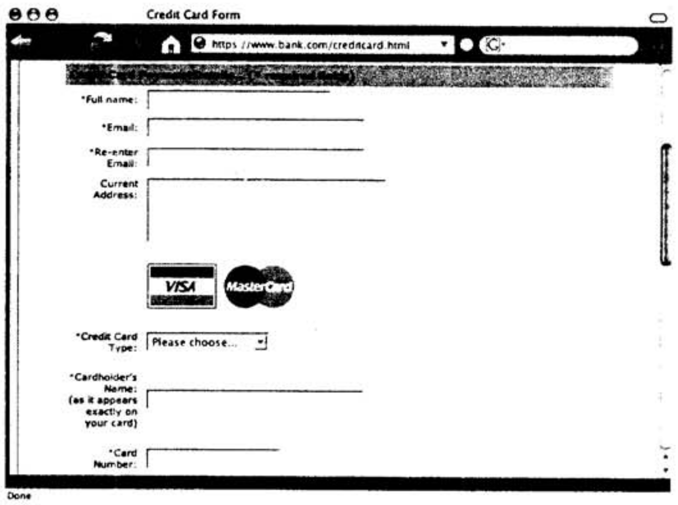


Figure 9.17 In Dynamic Security Skins, the trusted password window displays the visual hash pattern that matches the pattern displayed in the website window border.



*Full name:	
*Email:	
*Re-enter Email:	
Current Address:	
*Credit Card Type:	Please choose...
*Cardholder's Name: (as it appears exactly on your card)	
*Card Number:	

Figure 9.18 In Dynamic Security Skins, the browser displays the visual hash as a border around the authenticated website. If this pattern matches the pattern shown in the trusted window, the user can trust that this is the correct website.

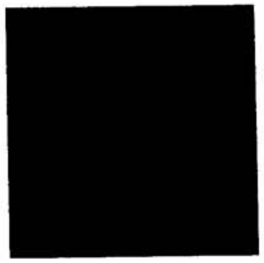


Figure 9.19 Visual hash generated independently by browser and server.

We suggest another approach in which we randomly generate images using visual hashes. As a visual hash algorithm, we use Random Art [2], which has previously been proposed for use in graphical password user authentication [40, 16]. 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 web page 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 cannot be mistaken for authenticated windows.

The approach of displaying the visual hash in the window border has some weaknesses. First, there are several ways for servers to override the display of window 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 web page without the browser chrome (a web page 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 web page 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.

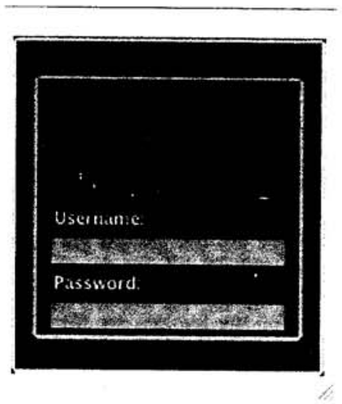
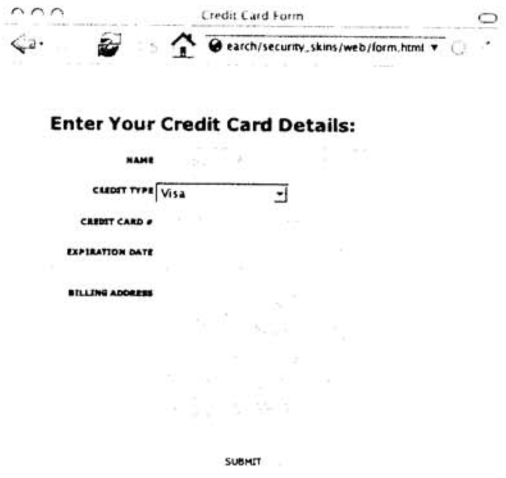


Figure 9.20 In Dynamic Security Skins, the trusted password window displays the visual hash that matches the website background.



Enter Your Credit Card Details:

NAME	
CREDIT TYPE	Visa
CREDIT CARD #	
EXPIRATION DATE	
BILLING ADDRESS	

SUBMIT

Figure 9.21 In Dynamic Security Skins, the browser displays the visual hash as the background of a form element. If the pattern matches the pattern in the trusted window, the user can verify that the request for information is from a known party.

Server-Generated Random Images. We now 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 web page 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 web page. 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 9.21. 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.

9.5.4 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 setup 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 cannot be tricked into providing it to a rogue site.

At each login, the bank website will trigger the browser to launch the trusted window. 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 is 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 with the bank server. If authentication is successful, the user is able to connect to the bank, and the trusted window will display the pattern image that it expects to receive from the bank.

The bank can use the pattern as a security indicator to help the user distinguish pages that have been authenticated and where extra caution is required (e.g., where sensitive information is being requested). For this to be an effective security technique, websites must establish standard practices for displaying the visual hashes and habituate users to expect their presence during security critical operations.

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 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.

9.5.5 Security Analysis

We now discuss the vulnerability of our scheme to various attacks.

Leak of the Verifier. The user's verifier is sent to the bank only once during account setup. 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 cannot 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 cannot be tricked into providing it to a rogue site.

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 cannot 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.

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 above, we have specifically designed our windows to make this type of attack very difficult to execute.

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 [14]. 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.

Spoofing the Visual Hashes. If this system were widely adopted, we expect that phishers will place false visual hashes on their web pages 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.

Public Terminals and Malware. A user can log in from any location with the browser extension installed, by supplying his password. However, a user cannot 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 provides protection against pharming attacks, where the users DNS hosts file is altered or where cache poisoning is used to redirect users to rogue websites. Even if users are misdirected to a rogue website and the user enters his password into the trusted window, the rogue website will not be able to capture any useful 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.

REFERENCES

1. Microsoft Developers' Network documentation. Platform SDK: Authentication. `http://msdn.microsoft.com/library/en-us/secauthn/security/gina.asp`.
2. Andrej Bauer. Gallery of Random Art. `gs2.sp.cs.cmu.edu/art/random/`.
3. Anti-Phishing Working Group Phishing Archive. `http://www.antiphishing.org/phishing_archive.html`.
4. M. Bellare and P. Rogaway. The AuthA protocol for password-based authenticated key exchange. Contribution to IEEE P1363.2, March 2000.
5. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
6. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Press, May 1992.
7. Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 244–250, New York, NY, USA, 1993. ACM Press.

8. A. Brusilovsky. Password authenticated Diffie–Hellman exchange (PAK). Internet Draft, October 2005.
9. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th Annual Symposium on Theory Of Computing (STOC)*, pages 209–218, Dallas, TX, USA, May 1998. ACM Press.
10. Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell. Client-side defense against web-based identity theft. 2004. 11th Annual Network and Distributed System Security Symposium (NDSS '04).
11. Cloudmark. Cloudmark Anti-Fraud Toolbar. `http://www.cloudmark.com/desktop/ie-toolbar/`.
12. Core Street. Spoofstick. `http://www.spoofstick.com/`.
13. Rohin Dabas, Adrian Perrig, Gaurav Sinha, Ting-Fang Yen, Chieh-Hao Yang, and Dawn Song. Browser enhancement against phishing attacks. Poster at Symposium on Usable Privacy and Security (SOUPS), July 2005.
14. Darren Davis, Fabian Monrose, and Michael Reiter. On user choice in graphical password schemes. In *Proceedings of the USENIX Security Symposium*, 2004.
15. Rachna Dhamija. Hash visualization in user authentication. In *Proceedings of the Computer Human Interaction Conference Short Papers*, 2000.
16. Rachna Dhamija and Adrian Perrig. Déjà Vu: A User Study. Using images for authentication. In *Proceedings of the 9th USENIX Security Symposium*, 2000.
17. Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the Symposium on Usable Privacy and Security*, 2005.
18. Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the Conference on Human Factors in Computing Systems*, 2006.
19. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542–552, New Orleans, Louisiana, 6–8May 1991.
20. Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.
21. Nathan Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Jospeh Konstan. Stopping Spyware at the Gate: A User Study of Notice, Privacy and Spyware. In *Proceedings of the Symposium on Usable Privacy and Security*, 2005.
22. Ralph Norman Haber. How we remember what we see. *Scientific American*, 222(5):104–112, 1970.
23. Helene Intraub. Presentation rate and the representation of briefly glimpsed pictures in memory. *Human Learning and Memory*, 6(1):1–12, 1980.
24. D. P. Jablon. Strong password-only, authenticated key exchange. Submission to IEEE P1363.2, September 1996.
25. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *Advances in Cryptology—EUROCRYPT ' 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 473–492, Innsbruck, Austria, 2001. Springer-Verlag, Berlin Germany.
26. Cynthia Kuo, Fritz Schneider, Collin Jackson, Donal Mountain, and Terry Winograd. Google Safe Browsing. Project at Google, Inc., June–August 2005.
27. T. Kwon. Summary of AMP (authentication and key agreement via memorable passwords). Submission to IEEE 1363.2, August 2003.
28. Micheal Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes. Pinceton University Press, 1996.

29. P. MacKenzie and R. Swaminathan. Secure network authentication with password identification. Submission to IEEE 1363.2, July 1999.

30. Philip MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Technical Report 2001/057, 2001.

31. A. J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

32. Microsoft. Erroneous VeriSign-issued digital certificates pose spoofing hazard. `http://www.microsoft.com/technet/security/bulletin/MS01-017.mspx`, 2001.

33. Steven Myers and Markus Jakobsson. Delayed password disclosure. Submitted to Financia Cryptography 2007, 2007.

34. National Fraud Information Center. Telemarketing Scams: January–June 2005. `http://www.fraud.org/telemarketing/tele_scam_halfyear_2005.pdf`, 2005.

35. Netcraft. Netcraft Anti-Phishing Toolbar. `http://toolbar.netcraft.com/`.

36. Out-law.com. Phishing attack targets one-time passwords. `http://www.theregister.co.uk/2005/10/12/outlaw_phishing/`, October 2005.

37. Oxford Information Services Ltd. Scam report. `http://www.millersmiles.co.uk/report/1722`, December 2005.

38. Bryan Parno, Cynthia Kuo, and Adrian Perrig. Phoolproof phishing prevention. In *Proceedings of International Conference on Financial Cryptograpy and Data Security*, February 2006.

39. Passmark Security. Protecting Your Customers from Phishing Attacks: An Introduction to Passmarks. `http://www.passmarksecurity.com/`.

40. Adrian Perrig and Dawn Song. Hash visualization: A new technique to improve real-world security. In *Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC)*, July 1999.

41. Princeton Survey Research Associates International. Leap of Faith: Using the Internet Despite the Dangers (Results of a National Survey of Internet Users for Consumer Reports WebWatch). `http://www.consumerwebwatch.org/pdfs/princeton.pdf`, October 2005.

42. Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. A browser plug-in solution to the unique password problem. Technical Report Stanford-SecLab-TR-2005-1, 2005.

43. RSA Security. RSA SecurID Authentication. `https://www.rsasecurity.com/node.asp?id=1156`, 2005.

44. R. Shepard. Recognition memory for words, sentences and pictures. *Journal of Verbal Learning and Verbal Behavior*, 6:156–163, 1967.

45. L. Standing, J. Conezio, and R. Haber. Perception and memory for pictures: Single trial learning of 2500 visual stimuli. *Psychonomic Science*, 19:73–74, 1970.

46. Hongxian Evelyn Tay. Visual validation of SSL certificates in the Mozilla browser using hash images, May 2004. Undergraduate Honors Thesis, School of Computer Science, Carnegie Mellon University.

47. Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (Second Edition)*. Prentice Hall, 2002.

48. J. D. Tygar and Alma Whitten. WWW Electronic commerce and Java Trojan horses. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996.

49. Visa USA. Verified by Visa. `https://usa.visa.com/personal/security/vbv/`.

50. Waterken Inc. Waterken YURL Trust Management for Humans. `http://www.waterken.com/dev/YURL/Name/`, 2004.

51. Alma Whitten and J. D. Tygar. Why Johnny Can't Encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the USENIX Security Symposium*, 1999.

52. J. Woodward. Security requirements for high and compartmented mode workstations. Technical report, MITRE Corp. MTR 9992, Defense Intelligence Agency Document DDS-2600-5502-87, Nov 1987.

53. M. Wu, R. Miller, and S. Garfinkel. Secure web authentication with mobile phones. In *DIMACS Symposium On Usable Privacy and Security*, 2004.

54. Min Wu, Simson Garfinkel, and Rob Miller. Users are not dependable—how to make security indicators to better protect them. Talk presented at the Workshop for Trustworthy Interfaces for Passwords and Personal Information, June 2005.

55. Thomas Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

56. E. Ye and S.W. Smith. Trusted paths for browsers. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, Aug 2002.

57. Zishuang (Eileen) Ye and Sean Smith. Trusted paths for browsers. In *Proceedings of the 11th USENIX Security Symposium*, pages 263–279, Berkeley, CA, USA, 2002. USENIX Association.

58. Ye Zishuang and Sean Smith. Trusted paths for browsers. In *Proceedings of the 11th USENIX Security Symposium,*. IEEE Computer Society Press, 2002.