BATTLE AGAINST PHISHING

Bharat Gautam[1], Mukesh Agrawal[2], Anishma Talwar[3], Rajan Jha[4]

1 Graduate Scholar, Computer Science & Engineering, JECRC Jaipur, Rajasthan, India, ***gautambharat.gautam@gmail.com***
2 Associate Professor, Computer Science & Engineering, JECRC Jaipur, Rajasthan, India, ***mukeshsir@gmail.com***
3 Lecturer, Computer Science & Engineering, JECRC Jaipur, Rajasthan, India, ***talwaranishma@gmail.com***
4 Lecturer, Computer Science & Engineering, JECRC Jaipur, Rajasthan, India, ***jharajan8@gmail.com***

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.**

There are 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.

Index Terms: **Phishing, Security Properties, Protection**

*****

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. 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. 2780 unique active phishing attack websites were reported in the month of March 2005 alone.
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].
We examine security properties that make phishing a challenging design problem. 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. We discuss related work.

2. TYPES OF PHISHING

2.1 Deceptive

Sending a deceptive email, in bulk, with a "call to action" that demands the recipient click on a link.

2.2 Malware-Based

Running malicious software on the user's machine. Various forms of malware-based phishing are:
Key Loggers & Screen Loggers
Session Hijackers
Web Trojans
Data Theft

2.3 DNS-Based

Phishing that interferes with the integrity of the lookup process for a domain name. Forms of DNS-based phishing are:
Hosts file poisoning
Polluting user's DNS cache
Proxy server compromise

2.4 Content-Injection

Inserting malicious content into legitimate site**.**
Three primary types of content-injection phishing:
Hackers can compromise a server through a security vulnerability and replace or augment the legitimate content with malicious content.
Malicious content can be inserted into a site through a cross-site scripting vulnerability.
Malicious actions can be performed on a site through a SQL injection vulnerability.

2.5 Man-in-the-Middle Phishing

Phisher positions himself between the user and the legitimate site.

2.6 Search Engine Phishing

Create web pages for fake products, get the pages indexed by search engines, and wait for users to enter their confidential information as part of an order, sign-up, or balance transfer.

3. SECURITY PROPERTIES

Paragraph Why is security design for phishing hard?[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:

3.1 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.

3.2 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.

3.3 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.
We revisit two properties proposed by Whitten and Tygar [8]:

3.4 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.

3.5 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.

4. SOLUTIONS

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

Solution is that we have to develop 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 will provide 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 easyto - 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).

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. 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.
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 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.

4.4 Secure Remote Password Protocol

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. 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), 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. 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 the SRP protocol is not enough to stop phishing attacks, because i t 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.

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.
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.

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

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. 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 web forms 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.

6. RELATED WORK

In this section, we provide an overview of the anti-phishing proposals. In general, attempts to solve the phishing problem can be divided into three approaches: third party certification and direct authentication, and phishing specific tools.

6.1 Third Party Certification

6.1.1 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. By displaying seals as graphics that can be easily copied, trusted seal programs ignore the “general purpose graphics” property.

6.1.2 Trustbar

The “Trustbar” proposal is a third party certification solution, where websites logos are certified. 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. Therefore, careful consideration must be given to the design of an indicator for insecure windows so that spoofed credentials can be easily detected.

6.2 Direct Authentication

Direct authentication approaches include user authentication and server authentication schemes.

6.2.1 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).

6.2.1.1 AOL Passcode

America Online’s Passcode has been proposed as a phishing defense. This program distributes RSA SecurID devices to AOL members. 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.

6.2.1.2 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 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.

6.2.2 Server Authentication Using Shared Secrets

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 and Verified by Visa, 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.

6.3 Anti-Phishing Tools

6.3.1 eBay Toolbar

The eBay Toolbar is a browser plug-in that eBay offers to its customers to help keep track of auction sites. 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.

6.3.2 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. 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, 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.

6.3.3 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. 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.

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