PhishGuard: A Browser Plug-in for Protection from Phishing

Yogesh Joshi[1], Samir Saklikar[2], Debabrata Das[1], and Subir Saha[2]
[1]IIIT-Bangalore, Electronic City, Bangalore, India.
{yogesh.joshi, ddas}@iiitb.ac.in
[2]Motorola India Ltd, Bagmane Tech Park, Bangalore, India.
{samir, subir.saha}@motorola.com

Abstract— ***Phishing is an act of identity theft aimed at acquiring sensitive information such as usernames, passwords, credit card detail etc., by masquerading as a trustworthy entity in an electronic communication. Phishers use a number of different social engineering mechanism such as spoofed e-mail to try to trick their victims. Data suggests that some of the phishing attacks have convinced up to 5% of their recipients to provide sensitive information to spoofed websites resulting in a direct loss of multi Billion Dollars across the countries. Though there are many existing anti-phishing solutions, Phishers continue to succeed to lure victims. In this paper, we have proposed a novel algorithm which aims at identifying a forged website by submitting random credentials before the actual credentials in a login process of a website. We have also proposed a mechanism for analysing the responses from the server against the submissions of all those credentials to determine if the website is original or phished one. Though our idea is generic and would work in any authentication technologies which are based on exchange of any credentials, our current prototype is developed for sites supporting HTTP Digest Authentication and accepting userid and password pair as credential. Our algorithm is developed within a browser plug-in for Mozilla FireFox v3.0. and can detect phishing attack conclusively.***

Keywords: Phishing, Security, Internet Theft, Browser Plug-in, HTTP

I. INTRODUCTION

Phishing continues to be one of the major threats for the current Internet [1, 2]. Unlike many other Internet threats which are also be mounted by casual people, Phishing is solely driven by direct financial greed and is mounted mostly by organized entities. This is obvious from the fact that all but two of the top 20 phishing targets were financial institutions [1]. Apart from this factor, Phishing attacks till recently are not based on some exploitation of weaknesses of the elements of Internet (i.e. Browsers or the network elements), but social engineering mechanism of fooling people to lure to sites of Phisher's choice to steal user credentials [3] making solution for phishing attacks much more difficult.

Recently at least one DNS vulnerability has been discovered which in essence can help Phisher to manipulate DNS data to divert user directly to any phishing site even though user actually has entered right URL in her browser [3]. This is possibly the first published technology weakness which a Phisher can exploit directly apart from their traditional tool of social engineering. Companies working on Internet Security domains were quick to prescribe the fix for this DNS weakness. We however, believe that more weaknesses are yet to be discovered which are open to exploitation by any Phisher. However, these technology exploits are short lived as counter measures evolve quickly and get distributed easily and reasonably quickly. Thanks to many research organizations and product companies who have been successful in doing this job of protecting Internet.

However, we believe that many more Phishers (at least those who are not so technology savvy) would continue to exploit the easy mechanism of exploiting human weakness through social engineering. The challenge in countering phishing attacks based on social engineering is that, user has to be guided to make the final decision to conclude if a site is phishing site or not. At least this is how the most of the current solutions are working [2, 5, 6].

Authors feel that with time, Phishers would become more intelligent and would start behaving like a "man in the middle" to increase the ferocity of their attack. In general there is not much technology challenge for Phisher to be a 'man in the middle' between the user and the original site that the Phisher is trying to mimic. For example, a user is lured to a phished site by some social engineer means (e.g. email with embedded link to the phishing site) which actually is supposed to be banking site for that user. The Phisher is already aware of the URL of this Bank site and hence can simply pipe the content of the original bank to the user. So now the Phisher is in between the user and his original bank and hence the Phisher can exactly mimic the banking site. So once user enters her name and password, the Phisher can submit the same in the original Bank site and grab the response page from the bank and push to the user.

We believe this requires the Phisher to be automata. In a recent paper by Samir et. al. [4] proposed a solution to stop this man-in-the-middle mediated Phishing attack. Their solution is based on using a modified CAPTCHA to remove the automata in between the user and the Phished site i.e., original bank site. In the same paper the author also touched upon another possible enhancement to detect normal (not the man-in-the-middle type) Phishing sites. Based on the fact that Phisher does not have user credentials and hence trying to steal the same. Ref. [4] gave a following possible suggestion, i.e., a user to

start with her login process may actually submit a random set of user name and password before submitting the right credentials. As the Phisher does not know which set of credentials is right it can not correctly responds with authentication failure or success. However, [4] does not talk about complete algorithm or implementation to achieve the same.

Building on the facts that till date most of the Phishing attacks are simple one and not the man-in-the-middle kind, we propose simple mechanism to detect phishing attack and provide protection from the same. Extending [4] we propose a browser enhancement exploiting the method of sending random credentials to test for presence of Phisher. Here we present the complete algorithm and have implemented the same in a browser which is able to detect Phishing sites successfully.

The rest of the paper has been presented as follows. Section 2 provides an overview of existing anti-phishing protections; Section 3 discusses the architecture of the algorithm. In Section 4 we describe the implementation of the plug-in in detail and the necessary client side and server side changes needed. Finally, Section 5 presents the vulnerabilities in the secured sites and how we can extend out approach to handle the same. Section 6 concludes the work with future vision.

II. Existing Anti-Phishing Protections

2.1 Spam Email Filters

Figure 1 explains how a user is tricked into a phishing attack. Majority of the phishing attacks start with an E-mail to the intended victims. The attacker creates the E-mail with the initial goal of getting the recipient to believe that the E-mail might be legitimate and should be opened. Attackers obtain E-mail addresses from a variety of sources, including semi-random generation, skimming them from Internet sources, and address lists that the user believes to be private [8].

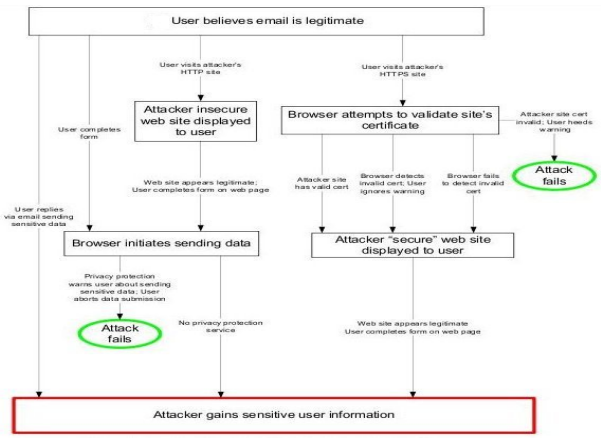


Figure 1: Phisher gains sensitive information [11].

As this core of the social engineering is email, obviously many of the Spam filters can actually cut off such email and hence phishing. However, as we know while spam filters are getting intelligent, the spammers are also coming with newer ways of bypassing such filters [3,7] making filter based solution ineffective. Apart from this, email is just one of the conduits of luring victims to phishing sites. For example voice calls, Chat rooms; Blogging sites can also lure victims equally well.

2.2 Browser Enhancements

There are some features in most browsers now to alert users of possible attacks. For example as shown in figure 1 that browser can detect if a site is non-HTTPS i.e. unsecured and user is submitting credentials in such page there by alerting the user of possible attack.

Also if the phishing site is secured, then the browser can check the certificate of the site and alert user of possible problem if the certificate is either not trustable or self certified. Apart from these, there is some work done by [5] introducing a secured password managers which automatically fills in user credentials in previously configured pages.

Ref [2] summarizes many other enhancements prescribed in trying to stop phishing attack. However, we see from the increasing ferocity of phishing attack none of these solutions have neither been successful nor been popular.

III. Proposed Solution

As described before, in this paper we propose a mechanism of detecting Phishing attack by submitting wrong credential in any login process. To start with, we will demonstrate this concept over HTTP Digest Authentication Scheme. We resort to Mozilla FireFox Browser and create a plug-in to demonstrate our idea.

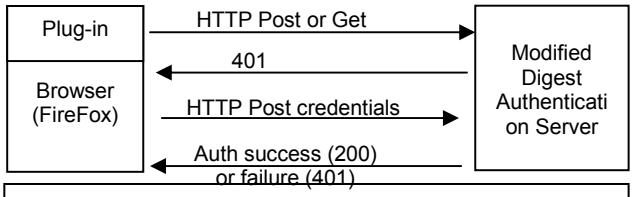


Figure 2: Architecture of end-to-end solution of our Anti-Phishing approach

3.1 Proposed Architecture and Algorithm

3.1.1 Architecture

Figure 2 captures the proposed architecture to demonstrate our idea. The proposed solution though built inside a browser as an add-on, there is slight modification in the authentication server needed to get better result. Our design consideration is that, user experience should be preserved and solution should work without involvement of user.

3.1.2 *Flow of Events*

3.1.2.1 *Client Side*

As we claim above, user experience is not changed; user tries to access a protected resource by clicking appropriate link from her browser. Once the browser prompts for user name and password, user inputs the same with whatever mechanism she prefers. For example, if the browser is capable of storing the credentials and helping to fill in on behalf of the user, then the user can continue to use the same method to input credentials in the login page. Once the credential is filled in, the browser waits for user to click to proceed with the authentication. Once user clicks appropriate button to continue the process, the browser plug-in takes the user submitted credentials and stores them in a temporary buffer. The browser plug-in then creates random set of credentials and posts it to the authentication server. For reason of simplicity, we believe that the password should only be randomized keeping the user name unchanged.

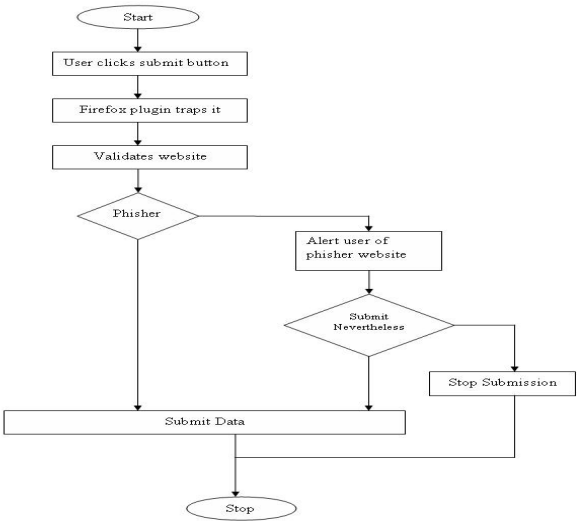


Figure 3: Flow of events for Proposed Anti-Phishing Solution

The authentication server has now received the random credentials as posted by the browser plug-in and responds with appropriate success or failure message. If the site is genuine one, which has original user credentials, must ignore all the random credentials posted and must respond with 401 indicating authentication failure. When the credential posted is correct one, the genuine site must respond with HTTP 200 OK allowing the user to continue to use the service.

If the site is a phishing site which does not have the original user credential at the maximum it can respond with randomly generated success or failure message. If the response is 'success', the browser plug-in can conclude the site is not the original intended site but a phished one. Now if the random response is 'failure', the browser plug-in can repeat the submission of the wrong password for few (random number of attempts) more times. If the Phisher continues to send random response (the number of attempts have been discussed in SubSection 3.1.3.1.1), it is easy for the browser plug-in to conclude that the site is phished one.

However if the Phisher decides to always respond with 'failure' irrespective of whatever credentials it receives, then after some random tries the browser plug-in must submit the right credential to which the Phisher ends up responding with "failure". So the browser plug-in now knows that the site is phished one. However, at this point browser plug-in should not stop posting random credentials as that would help Phisher to conclude that the last credential was the correct one. Based on this, our browser plug-in continues to post random credentials for few more times before discarding the session. In the mean time, the browser plug-in can indicate the user that it has concluded that the site with which the browser has a session is a phished one.

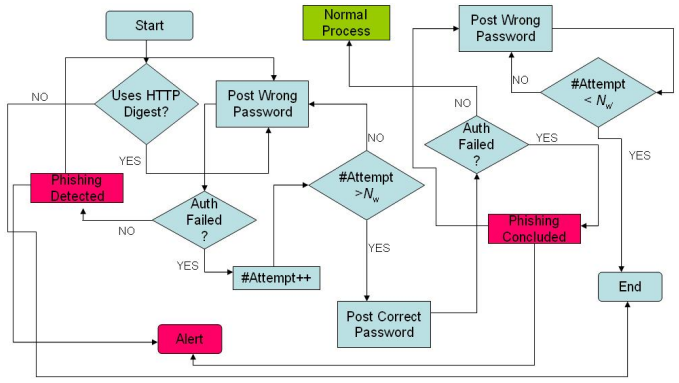
3.1.2.2 *Server Side*

As we suggested that our current version should work with majority of the secured websites, there is, in essence, no change in flow of events. The only change is that the server allows more number of authentication failures than typical 3 or so.

3.1.3 *Proposed Algorithm*

3.1.3.1 *Client Side*

Figure 4: Flow Diagram of Client Side Algorithm



In this case, once the user fills in the credentials, the browser plug-in stores that userid and password. Then it generates a random password which seemingly has similar

$R_F < N_W$, $R_S > 1$, Phisher with random response

$R_F > N_W$ and $R_S < 1$, User mistake or Phisher

look of the original password that user entered.

This new password along with the original userid is submitted to the site. Irrespective of response received, the plug-in continue to generate and submit few more passwords along with same userid. This is repeated for a random number of time say N_W where $N_W < N_{Max}$, and N_{Max} is the maximum number of allowed attempts of failure. After the N_W attempts with false password, the plug-in then submits the correct userid and password one time, i.e., $N_R = 1$, where N_R is number of times right credential is submitted.

Now user has received $N_W + 1$ number of responses. As we are using digest authentication, the responses from the server are either '200 OK' indicating success or '401 Unauthorized' indicating failure of authentication. If the number of success responses (R_S) are more than 1 or number of failure responses (R_F) are less than N_W we conclude the site as Phisher. If however, $R_F > N_w$ (i.e. received more failure response than expected), we have 2 possibilities; 1) the server always replies with failure irrespective of right or wrong credential and hence the site is Phisher, or 2) user has submitted wrong password or user-id or both.

Assume that user has entered correct password, we can now conclude the site is Phisher based on the above argument. The plug-in now can discard the session and indicate the outcome to the user. However, as the site is Phisher, it can conclude that the last password was the right one. In order to hide this fact, even though we are sure it is a phished site, we do not the discard the session after submission of correct password and continue to submit random password for another N'_W times where $N_W + N'_W >> N_{Max}$. However we already indicated that $Nw + 1 < Nmax$ otherwise user will block herself in an authentic site.

3.1.3.1.2 Handling User Mistake

This is a difficult problem and does not have very strong solution. We, however, propose that the browser plug-in may store all passwords user has for login into all sites of his interest after last successful login in each of the sites that he has visited. When the user types in a password in any login prompt, the plug-in checks if the password is new one and if it is new one then alert users that this is not a known password and hence may be incorrect one. So, at this time itself, it can be checked if the User has made a mistake or not. This would ensure that in the above verification process, the error condition always indicates that it is a Phishing site.

One can draw similarity between most of the password managers (including those which are built inside a browser today or those which run as standalone) with this proposal of ours and hence may argue that our solution brings in all the weakness of such password managers. However such claim is not true. The known weakness of a password manager is that it stores (in local machine) hash of the userid/password against some information about the site (e.g. URL) such that credentials can be fetched and filled in for all sites for which user agrees. This makes this local storage as the central point of vulnerability and attracting attackers. A single access to that database can jeopardize all user interest.

However in our solution we do not store anything other than all passwords and hence even if the attacker gets access to such information, he can not figure out the matching userid or the relevant site for which that credential is needed. Also in our case, many passwords are stored and hence loss of entropy for each password is further less making them virtually useless for the attackers.

3.1.3.1.3 Attack Analysis

It is clear that even if a user reaches a phishing site and uses our solution, she leaves $N_W + N'_W$ number of random password along with 1 correct one with the Phisher. The value of N_w could be any random value less than N_{max}. The reason it has to be random is so that we don't leave any conclusive value at the Phisher about N_w. N'_w on the other hand can take any large value, as it is being submitted to a Phishing site after the correct password, the higher this value the less probability of the Phisher being able to guess the correct password. We also have proposed that random passwords are generated in such a manner that visual or other ways of inspection would not be able to distinguish the right one from set of the wrong passwords.

3.1.3.1.4 Right Value of N_{Max}

Let us now see what the right value for N_{Max} is, i.e., the maximum number of authentication failure allowed in the server before locking out the user account. As the probability of detecting a correct password from the list of all passwords that Phisher has captured, is inversely proportionate to the $N_W + N'_W + 1$, total number of passwords submitted. On the contrary, a large value for N_{Max} increases the latency in login and computational burden in server. We believe that a user study is needed to see acceptable limit latency allowed along with server performance burden analysis to decide the N_{Max}.

Note that we have left $N_W + N'_W + 1$ number of passwords with the Phisher. Only one of those is correct. So now the Phisher can submit one by one password from list with userid in to the phished site (i.e. original site) till the right one leading to access to the site. This essentially defeats the whole purpose, since we assume the user would have changed the password after being alerted by the plug-in about the Phishing site.

3.1.3.2 Server Side

Assuming the server implementing HTTP Digest Authentication, we do not see any algorithmic change in server.

IV. IMPLEMENTATION

First we like to highlight that the choice of browser on which we want to build is decided by the fact that it is open platform and allow writing plug-in easily. Once we prove the concept, we would focus to do the same implementation over Microsoft Internet Explorer as this has huge market penetration and hence our solution can help many more users.

Apart from this we have decided to implement a test server which supports just HTTP Digest Authentication. This will allow us to implement our own algorithm to set the maximum number of authentication failure allowed.

4.1 Plug-in Design

The plug-in we have developed enables us to capture the HTTP request and tamper it before submitting so that we can meet our objective of identifying a Phishing site. This plug-in is written in Javascript and XUL (XML User Interface Language). XUL pronounced as "zool" relies on multiple existing web standards and technologies, including CSS, JavaScript, and DOM. The functioning of the plug-in can be seen in two steps, in the first step we trap the HTTP response to check if the site uses HTTP Digest, if so then we proceed with the second part of submitting random credentials and analyzing the response for the same.

To trap the HTTP response we use Javascript and some internal interfaces of Firefox. On installation of the plug-in every time a request is made by the browser, the response to the above request is captured to check if the request has been made to a web site requiring authentication. Once we successfully capture the response we then check the status line of the response. The status line consists of the HTTP version, Status code and the Reason-Phrase. Out of these three parameters, only status code is of our interest. The status code 401 indicates the website we are accessing uses authentication. The usual practice of the browser on receipt of this status code is to pop up a dialog box to the user, to fill in his credentials. Since we have to check this site for Phishing we now proceed with the second part of posting wrong credentials asynchronously.

We use AJAX (Asynchronous Javascript and XML) to post asynchronous request with wrong credentials. The wrong credentials are sent to the web site random (N_W) times. Response for these requests is checked for status code. The status code 401 indicates authentication failure and 200 indicates authentication successful. If it is a normal Phishing site then there are two possibilities, (i) the site always responds with authentication failure (401); (ii) the site always responds with authentication success (200). After N_W attempts of wrong credentials we then send correct credentials asynchronously and check for the response. So far we have sent $N_W + 1$ requests to the website. Thus, if the status for all $N_W + 1$ requests are 200 then we conclude that it is a Phishing site and alert the user. However, if we have the status as 401 for all $N_W + 1$ attempts, then there is a possibility that the user has entered wrong password or the Phishing site responds with 401 for all attempts. To make sure that user hasn't entered wrong password every time a user submits a password we store the hash of the entered password in a password file. We can thus cross check the hash of the current password with the hashes in the password file and look for a match. If there is a match then we conclude that the entered password is correct and the current site is Phishing thereby alerting the user. However, if there is no match we alert the user of a wrong password entry. Once the user corrects the password the same process is repeated to check for presence of Phishing. After concluding if the site is Phishing or not we further submit wrong credentials for N'_W times so that the Phisher doesn't conclude that the last credentials are the correct ones.

4.2 Server Side Changes

In our current version, the only change we expect to suggest for the server side is to increase the maximum number of trials with wrong password before a user account is blocked.

V. PHISHING ATTACK ON A SECURED WEB SITE

It is well known that HTTP Digest Authentication [10] remains an extremely rare implementation. In particular, none of the secured sites (e.g. Banking or other service providers) are using digest. The Digest has lived its life. Almost all secured sites today are using HTTPS [[9]] and then using some propriety implementation of standard userid/password based authentication mechanism. HTTPS uses either SSL or TLS as transport and establishes a secured pipe between the browser and the server. Once the secured connection is established, everything between the server and the browser is encrypted (as suggested by SSL/TLS protocols) and hence difficult for a Man-in-middle to know what is being carried. For example, a bank may first expose an unsecured page to user. HTTPS URL of the secured page may be embedded in that page. Once user clicks the secured link, the browser first starts a secured session and then exposes a new page asking for user credentials for login process.

As explained earlier predominant phishing attacks happen through social engineering process. So when a victim clicks the embedded link in an email that she has received, urging her to quickly change her userid and password in her bank account. So the user is already trapped and attracted to a phishing site. As we see it does not matter whether the phishing site is secured or not. In case the phishing site is unsecured and user submits credential, the current browsers are capable of alerting user. But even then many users get panicked and commit mistake by ignoring the alerts. In essence, secured sites are equally vulnerable to phishing attacks.

5.1 Extending PhishGuard! for Secured Site

The challenge in extending our solution for secured (i.e. HTTPS) site using userid/password as credential is that the response to authentication failure is sent in "200 OK" along with a new page with appropriate information alerting user of authentication failure.

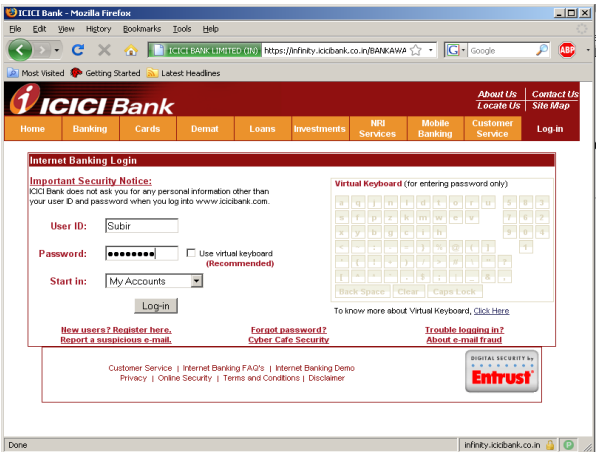


Figure 5: Secured Login Page of ICICI Bank

Figure 5 shows a secured login page of a famous bank in India with appropriate HTTPS URL. Figure 6 shows the response page after wrong credentials are submitted through the login page at figure 5.

As we see in the figure 6, it is difficult for automata to conclude (e.g. our plug-in) if the response in essence is success or failure of the authentication. Work is currently in progress to understand and propose a systematic mechanism of analyzing the responses from such sites to conclude if authentication is successful or failure. Result of these works would be published in another paper in near future.

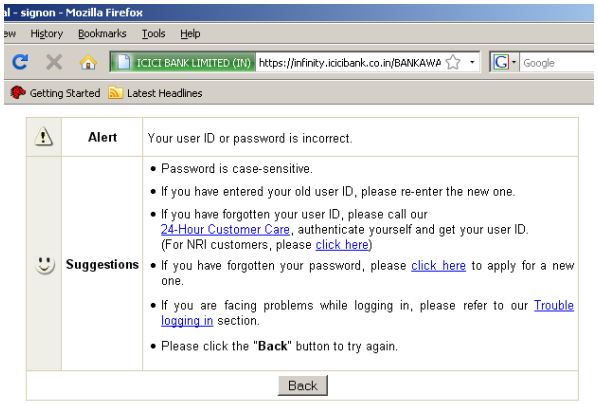


Figure 6: Authentication failure response

There is at least one simple solution for extending our plug-in to work for such sites. We believe much like digest authentication method, if such site agrees to respond with a standard message, our plug-in can be of help. With current focus being to come with a solution which does not need any major change in server side, we plan such solution for future work.

VI. Conclusion

In this paper we have proposed a new method of detecting phishing attacks by extending Firefox browser with a plug-in. We have presented a detailed algorithm and shown that our solution makes phishing attack a statistically probabilistic even after the user is successfully lured to a phishing site leading to leakage of credentials. This should be contrasted with current situation where once user reaches a phishing site, she becomes a confirmed victim. We have also developed a prototype as a browser add-on for Firefox with our proposed algorithm, giving successful results for HTTP digest based authentication site.

Acknowledgment

The authors thank Motorola India and IIIT-Bangalore for supporting this work.

VII. References

[1] IBM Internet Security Systems, X-Force 2008 Mid-Year Trend Statistics (http://www-935.ibm.com/services/us/iss/xforce/mid year report/xforce-midyear-report-2008.pdf)

[2] MessageLabs Intelligence: July 2008 (http://www-935.ibm.com/services/us/iss/xforce/midyearreport/xforce-midyear-report-2008.pdf)

[3] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, pages 581–590, New York, NY, USA, 2006. ACM Press.

[4] CERT Vulnerability Note VU#800113 (http://www.kb.cert.org/vuls/id/800113)

[5] Samir Saklikar and Subir Saha, Public Key-embedded Graphic CAPTCHAs, IEEE-CCNC 2008, Las Vegas, USA

[6] Passpet: Convenient Password Management and Phishing Protection Ka-Ping Yee and Kragen Sitaker, Symposium On Usable Privacy and Security (SOUPS) 2006, July 12–14, 2006, Pittsburgh, PA, USA

[7] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic Security Skins. In Proc. 2005 Symposium on Usable Privacy and Securtiy, pages 77–88, 2005.

[8] CNET – figure 1 taken from this reference

[9] IETF RFC 2818: HTTP Over TLS, http://www.ietf.org/rfc/rfc2818.txt

[10] IETF RFC: HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt

[11] Gregg Tally, Roshan Thomas and Tom Van Vleck, Anti-Phishing:Best Practices for Institutions and Consumers