Evaluating a Trial Deployment of Password Re-Use for Phishing Prevention

Dinei Florêncio and Cormac Herley
Microsoft Research, One Microsoft Way, Redmond, WA

ABSTRACT

We propose a scheme that exploits scale to prevent phishing. We show that while stopping phishers from obtaining passwords is very hard, detecting the fact that a password has been entered at an unfamiliar site is simple. Our solution involves a client that reports Password Re-Use (PRU) events at unfamiliar sites, and a server that accumulates these reports and detects an attack. We show that it is simple to then mitigate the damage by communicating the identities of phished accounts to the institution under attack. Thus, we make no attempt to prevent information leakage, but we try to detect and then rescue users from the consequences of bad trust decisions.

The scheme requires deployment on a large scale to realize the major benefits: reliable low latency detection of attacks, and mitigation of compromised accounts. We harness scale against the attacker instead of trying to solve the problem at each client. In [13] we sketched the idea, but questions relating to false positives and the scale required for efficacy remained unanswered. We present results from a trial deployment of half a million clients. We explain the scheme in detail, analyze its performance, and examine a number of anticipated attacks.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection—*Authentication, Unauthorized access, Phishing*

General Terms

Phishing

Keywords

passwords, phishing, authentication, access control

APWG eCrime Researchers Summit, 2007 Pittsburgh, PA, USA.
.

1. INTRODUCTION

Phishing for user credentials has pushed its way to the very forefront of the plagues affecting web users. An excellent review of recent attacks [22] shows the explosive growth of the phenomenon. The problem differs from many other security problems in that we wish to protect users from themselves: by social engineering users are manipulated into divulging their password. Two obvious approaches are to:

- Prevent the user from divulging the password
- Prevent the phisher from gaining anything useful from the password.

Preventing the user from divulging the password is extremely difficult. It requires first determining that a particular site is phishing, and then blocking data transfer (or at least password transfer) to the site. False positives (where we wrongly block a legitimate site) are completely unacceptable and false negatives (where we fail to detect a phishing site) are very expensive. So, if we wish to block the connection, this is an extremely unforgiving classification problem. We will review several anti-phishing technologies in Section 2, but many avoid this problem by issuing the user with a warning rather than blocking the information transfer. Thus, instead of preventing the user from divulging the password, they present the user with better, or more overt information, and hope that the user will act on it. Here, there are two major problems. First, to be useful, the warning must be issued *before the user types the password* (Javascript websites can transmit the password a key-at-a-time as it is typed). Thus when deciding whether or not to warn, the browser has little other than the URL and the actual document downloaded. We treat these approaches in more detail in Section 2, but we believe this to be an essentially impossible task (unless the client receives blacklist information from elsewhere). Blacklist approaches and their problems are tackled in Section 2.3.1 and [12]. Second, even when a user is presented with a warning, recall that she must *observe, understand and act on it.* If the user ignores the warning, nothing is accomplished and the phisher succeeds. Unfortunately, there is growing evidence that users suffer from "warning fatigue." The weakness of warnings as a tool in the particular case of phishing is shown in [28].

The great difficulty of getting users to act on security indicators was thoroughly documented recently by Schecter *et al.* [25]. Finally, of course, any warning that is generated by client-side information will be seen by the phisher before he launches the attack. It must be expected that competent attackers will endeavor to avoid triggering a warning that might decrease his yield.

In [13, 11] we sketched an approach that represents a considerable departure from previous efforts. Rather than attempt to prevent information leakage, we try to save users from the consequences of bad trust decisions. The scheme does so so without altering current password habits or infrastructure. Rather than stop passwords from being stolen our scheme seeks to make it quick and easy to identify when a password has been stolen, and simplify the task of mitigating the damage. Recall that, while it is too late to warn the user after she has typed the password, it may not be too late to prevent the phisher exploiting the account. If we considered each user alone this would still be a difficult task. Since users share passwords among sites all the time observing a user typing a password at an unfamiliar site is not actionable. However, by aggregating the information across many users we can build a far more reliable indication of a phishing attack.

The power of the scheme derives from scale, but this also implies a weakness: without a large enough deployment the aggregation of data is insufficient to allow low latency detection of sites. Thus [13] merely suggested the potential of the approach. We now have data from a trial deployment of over half a million clients. This allows us to evaluate the outstanding issues, such as false positives, server load, distribution of victims and the scale needed for a full deployment to be successful.

In the next section we examine related and previous work. Section 3 covers the scheme we are proposing in detail. Section 4 presents the results of a trial deployment of a version of the client to over half a million clients. Section 5 deals both with current phishing attacks, and the types of attacks that might evolve and how our system handles them. We also address the questions of whether the system can be used to mount DoS attacks.

2. RELATED WORK

In its relatively short history the problem of phishing has attracted a lot of attention. Broadly speaking, solutions divide into those that attempt to filter or verify email, password management systems, and browser solutions that attempt to filter or verify websites.

2.1 Email and Spam Solutions

The email that induces a user to enter her credentials at a phishing site is a particular kind of spam. Machine Learning tools for filtering have achieved a lot of success against spam in recent years[26], and many phishing emails get caught as spam. However, the task is complicated by the fact that the phishing email purports to come from a trusted institution, and purports to contain very important information; so it is unlikely that spam filtering tools alone will solve the problem

Since spoofing the email origin is an important part of the phishing attack a number of approaches seek to verify either the email path or the sender. Today it is trivially easy for a phisher to make an email appear as though it comes from `accounts@bigbank.com` and the goal of verification systems is to make that difficult or impossible. For example, Yahoo's Domainkeys proposal [7] proves the path of an email by having the originating mail server sign the message, and having that server itself certified by a DNS server. Adida *et al.* [5] also propose a trust architecture that allows detection of spoofed emails. However, it is a lightweight architecture and doesn't require a full public-key infrastructure (which is the main draw back of systems such as DomainKeys).

2.2 Password Management Systems

An early password management system proposed by Gaber *et al.* [14] used a master password when a browser session was initiated to access a web proxy, and unique domain-specific passwords were used for other web sites. These were created by hashing whatever password the user typed using the target domain name as salt. While the work of [14] predates the recent surge of phishing attacks, the idea of domain specific passwords is a very powerful tool in protecting users.

Several one time password systems exist that limit the phisher's ability to exploit any information he obtains. SecureID from RSA [1] gives a user a password that evolves over time, so that each password has a lifetime of only a minute or so. This solution requires that the user be issued with a physical device that generates the password. One time passwords can be based on an SKEY approach [27]. This solution requires considerable infrastructure change on the server side, and has not seen any significant deployment to general users. Several other approaches that involve changing the mechanism by which users are authenticated exists. For example two-factor schemes might effectively eliminate the phishing phenomenon. Oorschot and Stubblebine [21] propose authentication via a personal hardware device. In spite of the attractiveness of such schemes password authentication appears likely to persist.

Ross *et al.* [23] propose a solution that, like [14], uses domain-specific passwords for web sites. A browser plug-in hashes the password salted with the domain name of the requesting site. Thus a phisher who lures a user into typing her BigBank password into the PhishBank site will get a hash of the password salted with the PhishBank domain. This, of course, cannot be used to login to BigBank, unless the phisher first inverts the hash. Their system has no need to store any passwords. To prevent browser scripts from confusing the plug-in that a password is being typed, they use a key tracker and perform a mapping on the keys which is undone only when the data is POSTed. The user must prefix the password by typing a special control sequence, so a change of user behavior is needed. Further, different sites have different password rules (*e.g.* numeric, alphanu-

meric, *etc.*) and length requirements. These rules must be tabulated for the client.

Halderman *et al.* [16] also propose a system to manage a user's passwords. Passwords both for web sites and other applications on the user's computer are protected. In contrast to [23] the user's passwords are stored in hashed form on the local machine. To avoid the risk of a hacker mounting a brute force attack on the passwords a slow hash [20] is employed.

PassPet [30] by Yee and Sitaker also acts as a password management system implemented as a browser plugin. It generates unique passwords for each site and allows automated entry of credentials. As with many client password managers roaming is a problem. WebWallet [29] by Wu *et al.* warns users if it detects that credentials are being submitted to a non-trusted site.

2.3 Filtering Web-sites

A number of browser plug-in approaches attempt to identify suspected phishing pages and alert the user. Chou *et al.* [6] present a plug-in that identifies many of the known tricks that phishers use to make a page resemble that of a legitimate site. For example numeric IP addresses or webpages that have many outbound links (*e.g.* a PhishBank site having many links to the BigBank site) are techniques that phishers have used frequently. In addition they perform a check on outgoing passwords, to see if a previously used password is going to a suspect site. Earthlink's Scamblocker [2] toolbar maintains a blacklist and alerts users when they visit known phishing sites; however this requires an accurate and dynamic blacklist (see Section 2.3.1). Spoofstick [3] attempts to alert users when the sites they visit might appear to belong to trusted domains, but do not. Trustbar [17] by Herzberg and Gbara is a plug-in for FireFox that reserves real estate on the browser to authenticate both the site visited and the certificate authority.

Dhamija and Tygar [8] propose a method that enables a web server to authenticate itself, in a way that is easy for users to verify and hard for attackers to spoof. The scheme requires reserved real estate on the browser dedicated to userid and password entry. In addition each user has a unique image which is independently calculated both on the client and the server, allowing mutual authentication. A commercial scheme based on what appear to be similar ideas is deployed by Passmark Security [4]. The main disadvantage of these approaches is that sites that are potentially phishing targets must alter their site design; in addition users must be educated to change their behavior and be alert for any mismatch between the two images.

2.3.1 Problems with Blacklists and Filtering

Some anti-phishing solutions, such as [2], are based on keeping up-to-date blacklists, i.e., a comprehensive list of phishing sites. Apart from the difficult in populating the list, the required traffic can become unmanageable [12]. If the list is to be periodically broadcast to the users, the entire installed base must be reached, and clients are still vulnerable to new sites until the next download. Alternatively, if the list sits in the server, every client may need to contact the server every time it loads a page. For a comparison of the server load of our scheme *vs.* blacklist schemes see Table 1 of Section 4. In addition it bears mentioning that most current blacklists seem to be compiled using large amounts of manual or semi-manual effort. Users are often encouraged to report phishing sites (see *e.g.* [22]), but parsing these reports into useable lists requires much effort.

If the client attempts to identify a phishing site based on examining the page, smart design can obfuscate which link the user is actually looking at, by spreading the page across many domains. Further, the question of *when* the client performs its filtering becomes a serious problem. If it waits for the document to finish loading, a phisher can evade detection by including a single broken link (*i.e.* in Internet Explorer the onDocumentComplete event would not occur). If the client waits a fixed time interval, the phisher can delay loading the most suspicious elements of the page until after the checks are performed. Finally, a very interesting study by Wu [28] points out that users tend to ignore warnings. The study attempted to measure whether users notice the warnings provided by toolbars *even when the toolbar correctly detected the attack.* A large percentage of the participants did not change their behavior based on the warning.

2.4 Relation to our Method

A sketch of our basic approach was presented in [13] without analysis or data. While our scheme requires client code, we make no attempt to identify sites as suspicious based on their content or URLs. In fact we believe there may be no reliable way to distinguish phishing sites from other login pages based on examining the URL and HTML. What distinguishes a phishing site from other sites is not numeric IP addresses, or outbound links, or low pagerank, or lack of traffic or reputation information. Some or all of these characteristics are shared by many sites that are not engaged in phishing. What distinguishes a phishing site from most other sites is that the phisher requires victims to type a password. And not just any password: it must be a password that the victim has previously used at another site. Thus, we focus our attention on the one thing that a phisher requires: he must get the victim to type the desired password at the phishing site. So we regard the first instance of a password being re-used at an unfamiliar site as an interesting event. In that respect, we start with similar ideas to the outgoing password check described in [6]. We also populate a list of protected credentials by examining the data POSTed when the browser submits data. However in contrast to [6] we do not rely on the same approach to detect passwords typed at suspicious sites: we track the user's key entries and constantly check against known passwords. Once the password has been typed it may of course be in the hands of the phisher. However it is not too late to save that user's

account. We solve the problem associated with storing the information, in ways that related to the slow hash techniques presented in [16] and [20]. We avoid Javascript attacks using keyboard tracking, similar to the technique used in [23]. In contrast to [6, 23, 16], we do not attempt to detect an attack at each client, but rather rely on aggregating information at the server. Thus a phishing site using the online mock field password field that Ross *et al.* explain would be stopped by our scheme. It bears mentioning that each of the approaches [23, 6, 16] attempts to protect users individually, while our approach aims at detecting the attack globally.

Each of [23, 6, 3, 2, 17] relies upon a pop-up, or a traffic light type signal to alert the user to problematic sites. All of these schemes depend (in order to stop a phishing attack) on users changing their behavior in response to a warning. Our method for stopping the attack relies on the accumulation of information at the server and mitigation at the target. This is a strength, in that information from many users is aggregated. But it also a weakness, in that our approach requires that the client plug-in is used by a great many users in order to be truly effective.

3. OUR SCHEME

Our goal is to halt an attack in which a Phisher lures users to a website and asks them for userid (*uid*) and password (*pwd*) information. The architecture of our scheme consists of a client piece, a server piece and a backchannel to communicate with the target of the attack (*e.g.* BigBank, PayPal *etc.*). The client piece has the following responsibilities:

1. identify and add important credentials userid, and password to the protected list;
2. detect when a user has typed protected credentials into a non-whitelisted site;
3. report instances of 2. to the server.

The server has the responsibility of aggregating this information across users and determining when an attack is in progress. When it detects an attack it adds the phishing domain to a *Blocked list* and sends the hashes of the compromised userids accounts to the target domain with a view to initiating takedown and mitigation. Communicating with the target might appear non-scalable or to require much manual intervention and infrastructure. We show in Section 3.3.1 that this is not the case; it is actually simple and automatic.

Our client requires a full-featured browser that supports scripting (the defender's task is a great deal simpler if the browser does not support scripting languages). The trial deployment client (see Section 4) was implemented as an optional component of the [anonymized]Toolbar for Internet Explorer. It could equally be implemented as a plug-in for any other browser.

3.1 Client Role: tracking and reporting

We will explore the client's tasks in sequence. First, to produce a list of protected information; second, to detect that that information has been entered into another site; and, third, to report this to the server. Note that the client described in Section 4 differs in some respects from the ideal client we describe here. The trial deployment client implemented only features necessary to gather data to evaluate the scheme. It does not, for example, report hashed userids, and the server does not store any IP information.

3.1.1 Identifying credentials to be protected

The password, *pwd*, and userid, *uid*, are easy to identify on any page that uses HTML forms, and the browser of course knows the domain, *dom*, to which it just connected. We add $[dom, uid, pwd]$ to the protected list. Since it would not be safe to store the credentials in the clear, what we actually store in the protected list is:

$$P_0 = [dom, h(pwd), h(uid)],$$

where *dom* is the domain. We restrict *pwd* to be 16 characters long, and that we use salt that is specific to the client and to each table entry. Passwords estimated to have strength less than 20 bits were ignored.

Observe that we add P_0 to the protected list without knowing whether the login is successful or not. The `BeforeNavigate2()` event handler provided by Internet Explorer is used to do all of the above processing, *before* the HTTP POST data gets sent. As described, this would mean that if a user mis-types her password, a new entry (with the wrong password) would be generated in the protected list. We arbitrarily restrict the protected list to 256 entries; this should be more than enough to store all of the important password sites for any user, even if the list ends up containing many spurious entries. We employ a Least Recently Used strategy for maintaining the list. All entries of the protected list are stored using the Windows Data Protection API (the same mechanism used for storing passwords that are saved by the browser).

3.1.2 Detecting when protected credentials are typed

The near universal use of HTML forms makes populating the protected list relatively simple. It would be convenient if we could depend on phishers to use HTML forms; then we might just check the value of any password field submitted by a browser and see if it's on the protected list. It must be expected however that phishers will employ any means possible to conceal their intent from our defences. Using Javascript, for example, a phisher can present a page to the user that looks identical to the BigBank login page, but is code obfuscated in such a way that our plug-in cannot determine what data is being posted. An excellent account of several Javascript obfuscating techniques is given in [23].

To handle any and all tricks that phishers might employ we access the keystrokes before they reach the browser scripts. Figure 1 illustrates the procedure. For each key typed, we add the key to a FIFO buffer 16 characters long.

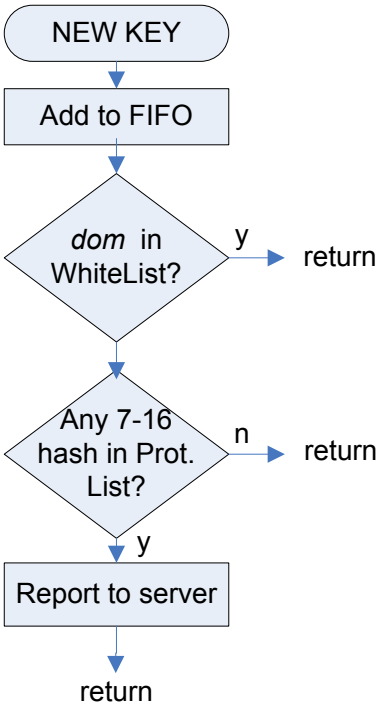

Figure 1: Keyboard analysis thread. Every key pressed gets added to a 16 character FIFO. Only if the hash of any of the last 7-16 characters matches the hash of a password and the domain is neither on the Cleared list nor Protected list is the server contacted.

Next we check to see which domain *dom* has control of the browser. If *dom* is in the whitelist, we do nothing (checking when credentials are to be added to the protected list is done in the separate thread detailed in Section 3.1.1). If *dom* is not in the whitelist, at each typed key we compute the hashes of possible passwords ending with the last typed character, and check against the appropriate hashes in the protected list. More specifically, since passwords can be between 7 and 16 characters, we need compute hashes of 10 strings. Since each entry in the protected list has a specific salt, we need to compute a total of $10 \times 256 = 2560$ hashes, and compare with the appropriate entries in the protected list. When one of the FIFO hashes matches a protected list H1 value, it means that the just-typed string matches a password on the protected list. Since we already determined that we are connected to a non-whitelisted domain this event is worth reporting to the server.

3.1.3 *Reporting to the Server*

Whenever a hit is generated, we inform the server. The report informs the server that a password from dom_1, on the protected list, was typed at dom_R, which is not on the whitelist. More specifically, what the client reports is:

$$C_{report} = [[(dom_1, h(uid_1)), (dom_2, h(uid_2)), \cdots], dom_R, h(IP)],$$

where $h(IP)$ is the hash of the IP address of the reporting computer. Recall that users commonly use the same password at several legitimate sites. We will see next that this is not a problem.

3.2 Server Role: aggregation and decision

The server has the role of aggregating information from many users. As detailed in Section 3.1.3, C_{report} is sent when protected credentials are typed into a non-whitelisted site; the server stores this record along with a timestamp. Recall that the server receives a report of every instance of a user typing protected credentials into a non-whitelisted site. In fact, it receives a vector with all legitimate domains that share that same password. The server collects this information for all non-whitelisted sites, and uses that to include the site on the blocked and white lists. We deem dom_A to be a phishing site that is attacking dom_B if:

1. dom_A appears in a list five or more times with dom_B
2. dom_B is in 75% of the SURL lists where dom_A appears
3. Number of logins at $dom_B \geq$ 5x number of logins at dom_A
4. dom_A not on "Whitelist"
5. dom_B is on a list of "phishable sites."

The simple Blocked list rule given suffices for the phishing attacks reported to date. However, the logic on the server might have to evolve as attacks evolve. One of the strengths of our system is that by making use of such reliable information (*i.e.* password re-use), aggregated across many users, the server is in a position to identify and stop an attack. To succeed with a distributed attack (see Section 5.1.4 and [18]) the phisher would have to make the pattern of client reports statistically insignificant.

3.3 Backchannel: notification and mitigation

A component common to many good security systems is that the tools and responsibility for mitigating the problem reside with the party most motivated to fix it. To this end a key element of our scheme is delivering to the target the information that it is under attack, the coordinates of the attacker, and enough information to identify the (possibly) compromised accounts.

3.3.1 *Notifying the Target*

When the server determines that an attack is in progress it must notify the institution under attack. There are two very important components to the information the server can now provide dom_R :

- The attacking domain *dom*
- The hashes h(*uid*) of already phished victims.

The mechanism for notifying dom_R that it is under attack is simple. An institution BigBank that wishes to have its domain protected must set up an email account `phishreports@bigbank.com`. Reports will be sent to that address. For verification purposes the email header will contain the time, the domain (*i.e.* "Bigbank") and the time and domain signed with the server's private key. Any email arriving at that address that does not conform to the protocol will be dropped on the floor by the BigBank mail server. In this manner any spam or other email that does not come from the server can be immediately discarded.

Additional victims are likely to be phished in the interval between the server notifying dom_R and the phishing site *dom* actually going offline. The server sends additional reports to the target as each of these arrive. These messages again are signed so that verification is simple, and thus h(*uid*) for each victim can be routed to dom_R without delay.

We point out that scale of deployment is a key factor here. Only if deployment of the client piece accounts for a significant percentage of web users will BigBank find it advantageous to set up the mail account necessary. The quality and timeliness of the information received depends on the scale of the deployment. Optimally the client should be implemented as part of a browser with large installed base such as Microsoft's Internet Explorer.

3.3.2 Mitigation

On receiving an attack report from the server dom_R can initiate actions to

- Takedown the attacking site *dom*
- Limit activity on the compromised accounts.

Web-site takedown is the process of forcing a site offline and can involve technical as well as legal measures. Several companies specialize in these procedures (*e.g.* Cyota, Branddimensions and Internet Identity). While "Cease and Desist" and legal measures are pursued, a simple denial of service attack can put the phisher out of commission. Indeed by flooding the faked login page hosted by *dom* with randomly generated *uid* and *pwd* fields it is easy to ensure that the phisher will have to trawl through much junk to find the few (*uid*, *pwd*) pairs from actual victims.

In addition the target can limit activity on compromised accounts. This does not necessarily mean that all access to the account is denied, or innocent features disabled. For example, if the target is a bank, then recurring payments, or bill payments to recipients already on record represents little risk. However payments to new recipients, or any attempt to change the address of record of the account should clearly be disabled.

4. RESULTS OF TRIAL DEPLOYMENT

As we make clear in [13], detection of phishing sites based on PRU relies on a very large deployment. For example, at a threshold of 5 reports, a 1% deployment would only detect a phishing sites after about 500 users from the population have fallen victim to the page. Most phishing attacks are much smaller than that (as the data will indicate), and therefore would go undetected by a 1% deployment of PRU.

Nevertheless, a number of questions are unanswered. In particular, questions relating to the false positive rate, and the willingness of the users to opt into the system, can be answered with a small deployment. Furthermore, reliable statistics about the size, distribution, and timing of phishing attacks, and about typical password re-use patterns, can help in evaluating the effectiveness of the problem and help foresee potential problems.

A trial deployment to capture the additional data necessary for the analysis was carried out. Our client software shipped as a component of [anonymized] toolbar. The component was optional, and users were presented with an opt-in agreement. The toolbar was first available for download on the web on 7/24/2006, and a total of 544960 clients installed and activated by 10/1/2006. The opt-in rate was greater than 60 % of those offered the client and was approximately the same as for other non-security related options.

4.1 Trial Implementation

The client consists of a module within the toolbar that monitors and records Password Re-use Events (PRE's). It contains the following main components.

HTML password locator: this component scans the document object model in search of filled-out password fields, and extracts the passwords. This search is initiated every time the browser `BeforeNavigate2` event occurs. Once the password is found it is hashed and added to the Protected Password List (PPL).

Protected Password List: This list contains the password hash and the full URL of the receiving server. All of the information in the PPL is stored using the Data Protection API (DPAPI) provided by Windows [24]. For security reasons weak passwords (with bitstrength < 20 bits) generate no entry in the PPL.

Realtime password locator: this component maintains a 16 character FIFO that stores the last 16 keys typed while the browser had focus. If any 7-16 character section of the FIFO matches a password in the PPL, it checks whether the URL of the current server is among the URLs previously associated with the password. If not a Password Re-use Event (PRE) report is sent to the server.

PRE Report: this contains: the current (primary) URL and all of the URLs previously associated with the password (secondary URLs). Observe that neither the password, nor its hash are sent in the report. There is no personally identifying information in the report. Comments on interesting and unexpected findings from that data are detailed in [10].

Privacy: A number of measures were taken to protect the privacy of those who opted in. No Personally Identifying Information was gathered from the clients. Neither passwords, nor their hashes were sent to the server. IP addresses from which reports were received were not stored at

the server. In addition the time at which PRE reports were received was timestamped at the server with granularity 10 minutes to make identifying users by login times difficult. Finally, the hashes of the userID were neither stored nor sent with the report [1]. A privacy audit was performed and published [19].

Server: The server records each received report and stores with a per-PRE report ID and a timestamp. It does not record any location information such as IP address that might allow for identification of the user or his/her location.

4.2 Data Analysis

Downloads began almost as soon as the client became available on the web, and data from the clients began to flow shortly thereafter. The data sheds considerable light on web users password habits. A detailed study of aspects of the data unrelated to phishing can be found in [10]. The 500K clients helped us answer a number of questions, and estimate performance of the algorithm.

4.2.1 Phishing is a Big Problem, Phishing is a Small Problem

This deployment is only a survey, as 500,000 clients correspond, to, maybe, about 0.01% of internet users. Yet, we point out that this is a big survey. For example, the results in section 4.2.2 are based on these 500k clients. Previous surveys on the size of the phishing problem by Gartner (2006) and the Federal Trade Commission [9], were based on samples phone or email samples of 5000 and 4057 people respectively. Thus this deployment represents 100 times more participants than previous surveys or studies of the problem. In addition, our deployment client measures what clients actually do, rather than what they remember and say they did. The Gartner study was based on an email and the FTC [9] on a phone survey.

Phishing is a big problem when measured by the standards of the volume of phishing email received by users, and by the number of new reported phishing sites. APWG [22] reports more than 37000 new phishing sites per month in late 2006. This is large by any standard, but it appears likely that a majority of these sites get few or no victims. For example if each received an average of 100 victims, and we assume an active web population of 500 million users, this would imply that $100 \times 3.7e6 \times 12/500e6 \approx 8.8\%$ of users were being phished annually. This seems higher than common sense would indicate.

4.2.2 Size and Distribution of Phishing Attacks

We had access to a list of phishing sites that were active during a three week period toward the end of the study. These sites were determined and verified to be phishing by a third party vendor. There was an average of 436 k clients during this three week period. We recorded 101 PRE reports listing one of the verified phishing sites as the primary URL. This implies that the client has typed at the phishing site a password previously used at another site on the user's PPL, which is a fairly good indication that the user has been "phished." We can use this to get an estimate of the annualized fraction of the population being phished as

$$\frac{101 \times 365}{436000 \times 21} \approx 0.00403.$$

Thus the data indicates that about 0.4% of the population falls victim to a phishing attack a year.

[1] In the future, if a full deployment relies on the back-end protection, hashes of the UserID would have to be stored in the PPL and sent with the reports. No password information is ever sent.

4.2.3 False Positives

A major outstanding issue is whether the algorithm used in Section 3.2 to decide dom_A is phishing dom_B produces many false positives. False positives will happen when the first few users visiting a new site have an overlapping set of password re-use sites. For example, if the first five visitors to `myNewStore.com` all use the same password at the store as they have used at BankOfAmerica, the server would decide that the site is phishing BankOfAmerica. In [13] our conjecture was that the password re-use habits among uncorrelated users would be diverse enough to keep the false positive rate very low. The trial deployment confirms that.

In fact, the diversity of sites is greater than we had anticipated. Our client population visited a total of 143k distinct login URLs. There were approximately 20 million total login events. If we applied only the first three requirements of the phishing detection in Section 3.2 (*i.e.* omit the whitelist and phishable tests) there were only 41 false positives among the 143k URLs. That is, the pattern of password re-use among our clients is such that less than 0.03% of sites accidentally trigger an alert *even before we apply any whitelist or phishable rules.* This confirms that phishing sites have a very different pattern of PRE reports from that of innocent sites, even when a suspicion threshold of five reports is used.

Since we did not have authoritative and exhaustive white or phishable lists applying the last two tests must be modeled. We approximate the first by whitelisting any URL for which we have login history more than 7 days old. (Note: we do not suggest this as a reliable whitelisting strategy; we merely assume that in the absence of active attacks on the trial deployment sites with history are good). Using this simple whitelist drops the number of positives from 41 to 12 (or 0.008% or URLs).

Since an exhaustive list of phishable (*e.g.* bank) URLs was not available, we settle for excluding instead cases where the target was among the 10 most visited non-phishable URLs. That is, we constructed a list of the 10 most trafficked sites (*e.g.* as `hotmail, myspace`) which are not usually targets of phishing attacks. If dom_A satisfied the first four criteria in Section 3.2, but dom_B was on our non-phishable list we discounted it. This reduced the number of positives (among the 143k distinct login URLs) to 2.

Given that we believe more exhaustive white and phishable lists can be built, we believe this false positive rate is negligible. In fact, we believe that with a more exhaustive

whitelist (based on traffic from a larger client base) it will be possible to drop the threshold from 5. We pursue that line of enquiry as a response to distributed attacks in Section 5.1.4.

4.2.4 Size Distribution of Attacks

We assume that not all phishing attacks are equally large. There were 60 distinct verified phishing sites among the 101 for which we received a PRE report in the three week period mentioned in Section 4.2.2. There is a certain distribution of reports across sites, *e.g.* we received 4 reports from one site, 3 from few, 2 and 1 from a great deal more. The standard method of estimating the probability mass associated with unobserved types from a sample is to take the Good-Turing estimate [15]. This gives that the probability mass of the missing types is $N_1/N = 33/101$, where N_1 is the number of sites with one report and N is the number of reports. We model the distribution of phishing sites as an exponential $e^{-\lambda x}$, where x is the site number, ordered by decreasing number of PRE reports (so $x = 0$ has most reports, $x = 1$ has second most and so on), and get $\lambda = 0.0257$. This accords well with the expectation that a few phishing sites get many victims, but there is a long tail which receive very few.

4.2.5 How many Victims can PRU Save?

The efficacy of our scheme depends on the distribution of victims among sites. If every single phishing site received fewer than 5 victims the detection of Section 3.2 would save nobody. We use the exponential distribution of victims from Section 4.2.4 to estimate the save rate. We normalize so that the total number of victims (integral of the whole distribution) matches the estimated number of victims over the population of 50 million.

$$c \cdot \int_0^\infty \lambda e^{-\lambda x} dx = 101 \times \frac{50e6}{436e3},$$

giving $c = 11.6k$. The number of victims that cannot be saved is all victims at sites that have fewer than 5 victims, and the first 5 at all other sites. Using the above model there will be

$$\lambda e^{-\lambda x_0} = 5 \Rightarrow x_0 = 205,$$

sites that have more than 5 victims (over the three week period), and then

$$ce^{-\lambda x_0} = 59 \text{ victims},$$

at sites that receive 5 or fewer victims. There will be at least $5 \times 205 = 1025$ victims at the larger sites before suspicion is raised. The total number of expected victims is

$$c = \frac{50e6 \cdot 101}{436e3} = 11.6k.$$

Thus, we achieve

$$1 - \left(\frac{205 \cdot 5 + 59}{11.6e3} \right) \approx 0.907,$$

or a rate of 91%.

	PRU Scheme	Contact Server for Blacklist	Broadcast Blacklist
Server hits/day	$1e6$	$1e10$	$1e8$

Table 1: Traffic comparison. Assuming 100 million deployed clients, each browsing 100 pages/day. We assume each client establishes four new accounts a year re-using an existing password. Observe that the PRU scheme traffic is far smaller than a blacklist scheme. In addition, if the blacklist is broadcast once a day, there is a large latency in the list.

4.3 Server Traffic

An important consideration of any web service is the resources it consumes, and how this can be expected to scale with deployment.We now estimate this. The scheme sends a message to the server only when a protected password is typed into a non-whitelisted site. It takes time for the Protected List to fill (*i.e.* for a user to visit all of her password protected accounts). Once this happens reports are sent only when a new account is established (or the user is phished). Figure 2 shows the number of sites sharing a password *vs.* the age of the client. As can be seen, after 50 days the average password is being used at 5.5 sites and is being added to about 0.1 new sites per month. We also measure (see [10]) that the average client has 6.5 passwords after 50 days. It is the incremental rate at which existing passwords are re-used at new login sites that determines the rate of PRE reports arriving at the server. Thus we expect 0.65 reports per client per month. At an installed base of 50 million users this would translate to less than one million server hits per day (a load that is minor by the standards of modern web services). Table 1 compares the server load from our scheme with that of a blacklist scheme.

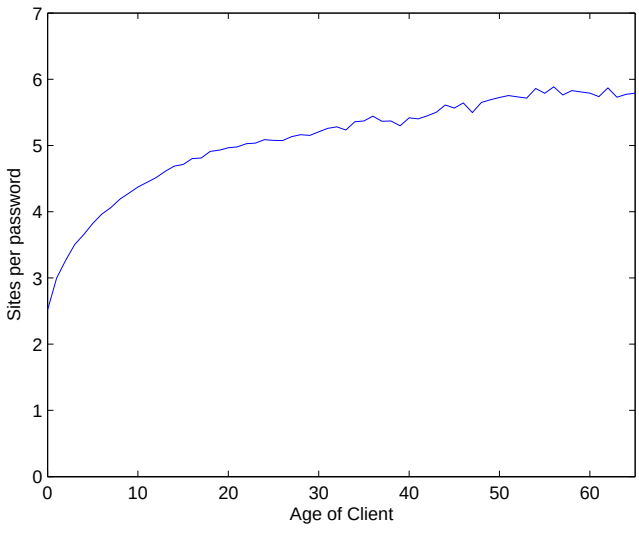

Figure 2: Number of sites per password *vs.* age of client in days. The average password appears to be used at about 6 different sites.

5. ATTACKS

There are two main ways of succeeding with a phishing attack on the system. First, the phisher may try to prevent clients from reporting to the server. Second, he may try to prevent the server from detecting an attack from the reports it receives by hiding below any suspicion threshold the server may have. Finally, a vandal may try to use the system to mount an denial of service attack at a legitimate site.

5.1 Preventing the client from reporting

We kept the logic of the client piece as simple as possible, so that it is hard to prevent it from reporting entry of protected credentials at non-whitelisted sites. This means that, once deployed, the client piece probably does not need to be updated as phishing attacks evolve. To make the client piece as durable as possible we have considered several attacks.

5.1.1 Flushing the protected list

A Phisher might try to circumvent the protection by removing some (or all) of the passwords from the protected list. For example, since the protected list has only 256 entries, a phishing site could submit (using HTML forms) 256 strings as passwords to a random site. As described in Section 3.1.1 this would effectively "flush" everything from the protected list because of the Least Recently Used maintenance rule. To avoid this attack, before accepting a new entry, from the HTML form data (as in Section 3.1.1), we match the password with the keyboard buffer, effectively requiring that the password have been actually typed at the site. It is unlikely that a Phisher can induce a victim to actually type hundreds of password-like strings.

5.1.2 Hosting on a whitelisted domain

A phisher might attempt to host on an existing, legitimate site. For example putting the phishing page up on an ISP member site, like members sites on AOL or MSN, or a small site like a community group association, or by employing a Cross-Site Scripting (CSS) attack. Each of these is handled by proper design of the client whitelist.

It is easy to handle ISP member sites by including the ISP, but excluding certain sub-domains from the whitelist. Small groups like community associations cannot be depended upon to prevent break-ins. Thus the client whitelist should contain only very large commercial domains. Recall that a site can be on a users protected list, without being on the whitelist. CSS attacks actually host the phishers page on a target domain. For this reason, only sites that can be depended upon to maintain basic levels of security should be permitted to populate the client's whitelist.

5.1.3 Tricking the user into mis-typing the userid

An earlier version of the algorithm hashed the combined *uid/pwd*. This provided a way a way for a Phisher to circunvent the system, by forcing the user to mis-type the *uid*. Normally, as you type your userid, the letters show up at the screen. Suppose the Phisher introduces a page with a script where the third character you type does not show up in the screen. You'd think you did not press hard enough, and would re-type the character. As you do that, the keyboard buffer will have that character twice, and it will not hash to the protected entry. We note that this attack is not possible with the password, since the password does not show up in the screen as you type (only ****). If something goes wrong, most users simply delete the whole thing and re-start.

5.1.4 Distributed Attack

A possible approach for a phisher who seeks to evade detection is to distribute the attack. Rather than phish BigBank by directing victims to PhishBank the phisher may use many domains, or numeric IP addresses. Thus when clients report sending their BigBank credentials, it fails to trigger an alert at the server. For this reason, we believe the server logic needs to adapt as phishers adapt. For example, while the destination for several BigBank credentials may be distributed, a large increase in the number of reports for a given whitelisted domain is in itself worthy of suspicion.

Recent data seem to point to evidence of an increase in this kind of distributed attack. We point out that, in the limit, attacks can be such that each victim is directed to a different URL. In this case, any schemes based on blacklisting, will become completely ineffective. It might appear that our approach also would suffer this fate. In Section 4.2.3 we suggested that the false positive rate was so low that we might consider dropping the threshold for suspicion from 5 PRE reports. In fact, with a large and reliable whitelist we suggest that it can be dropped to one. That is, if we receive a single PRE reporting that dom_A has accepted a password previously used at dom_B and dom_A is not on a whitelist we immediately inform dom_B, as suggested in Section 3.3.1. It might appear that for this we would have to whitelist every possible URL (which runs to billions of pages). Recall, however, that PRE reports come only from login URLs: *i.e.* pages that either have a password field, or that have accepted text that was previously typed into a password field on another site. First, there are many orders of magnitude fewer login pages than there are ordinary pages. Second it is only the new login pages that might trigger false positives. Third, by informing the suspected target we inconvenience only those users who attempt to login to the suspected target dom_B and conduct a suspicious transaction, before dom_B has had time to determine that the alarm is false.

5.1.5 Redirection Attack

Similar to the distributed attack is a Redirection attack where a phisher directs all victims to a single URL, but each victim is redirected to a different (possibly unique) address to be phished. For example the phishing URL might originally redirect to IP_1, but as soon as the first victim is caught it redirects to IP_2 and so on. This might appear to confuse our system by distributing the client reports one at a time

among many addresses. To circumvent this we include in the client report any URLs visited in the last minute. By intersection, the redirecting (phishing) URL can be detected.

5.2 Denial of Service on a Site

Related to false positives is the possibility of using the system to mount an Denial of Service attack on a site. Specifically, suppose someone falsely reports that MomnPop.com has accepted BigBank passwords enough times to cause the server to conclude that MomnPop is phishing BigBank.

First, note that we track the IP of the reporting client; a single client reporting the site repeatedly will have no effect. Second, no site that is on the client whitelist (of large institutions) or the server's much larger whitelist can ever be placed on the Blocked list. The most powerful defense, however, is that if MomnPop does get on the Blocked list the consequences of the false positive are almost undetectable as shown in Section 4.2.3 above.

5.3 Roaming and Internet Cafes

A user at an internet kiosk will not be protected. By this we mean that typing her BigBank credentials at a phishing site will generate no report to the server. This is so, since the user does not have access to their protected list. A main advantage of the scheme, however, is that by aggregating across many users the server can detect an attack much more quickly and the Block list can be expected to be updated with low latency.

6. CONCLUSION

We proposed a scheme which has the potential of neutralizing most phishing attacks. The client piece stores hashes of important personal information (e.g., passwords), and reports to the server whenever this information is typed at non-whitelisted sites. The server aggregates those reports to produce a very responsive blocked list. The scheme requires no change in user behavior. In fact, no notification is ever made to any user. Instead, hashes of the compromised accounts are passed onto the financial institution, which will proceed to take down the phishing site, and to limit on-line privileges at the compromised accounts. Restoring full access to the compromised accounts will follow each institutions procedure, but will likely be similar to the ones used today to restore access to a stolen credit card. False positives are extremely rare, and almost innocuous. The wrongly accused site is not impacted, and suffers no loss.

The extent of deployment is key in making the scheme effective. The more users involved in the scheme, the faster the detection. Additionally, the willingness of the financial institutions to put back channel mechanisms in place will be proportional to the number of customers reached. Deployment by one or more large browsers is at the same time a requirement for the scheme and a win for the browser's users. **Acknowledgements:** the authors wish to acknowledge enormous help and assistance from Steve Miller, Geoff Hulten, Anthony Penta, Raghava Kashyap and especially Steve Rehfuss. Cem Paya provided several suggestions and attacks that have materially improved the scheme.

7. REFERENCES

[1] `http://www.rsasecurity.com`.

[2] `http://www.scamblocker.com`.

[3] `http://www.spoofstick.com`.

[4] `http://www.passmarksecurity.com`.

[5] B. Adida, S. Hohenberger, and R. L. Rivest. Fighting phishing attacks: A lightweight trust architecture for detecting spoofed emails. *USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*.

[6] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. Mitchell. Client-side defense against web-based identity theft. *Proc. NDSS*, 2004.

[7] M. Delany. Domain-based email authentication using public-keys advertised in the dns. 2004. `http://www.ietf.org/internet-drafts/draft-delany-domainkeys-base-01.txt`.

[8] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. *Symp. on Usable Privacy and Security*, 2005.

[9] Federal Trade Commission. Identity Theft Survey Report. 2003. `http://www.ftc.gov/os/2003/09/synovatereport.pdf`.

[10] D. Florêncio and C. Herley. A Large-Scale Study of Web Password Habits. *WWW 2007, Banff*.

[11] D. Florêncio and C. Herley. Stopping a Phishing Attack, Even when the Victims Ignore Warnings. *MSR Tech. Report TR-2005-142*, 2005.

[12] D. Florêncio and C. Herley. Analysis and Improvement of Anti-Phishing Schemes. *SEC*, 2006.

[13] D. Florêncio and C. Herley. Password Rescue: A New Approach to Phishing Prevention. *Proc. Usenix Hot Topics in Security*, 2006.

[14] E. Gaber, P. Gibbons, Y. Matyas, and A. Mayer. How to make personalized web browsing simple, secure and anonymous. *Proc. Finan. Crypto '97*.

[15] W. Gale. Good-Turing Smoothing Without Tears. Statistics Research Reports from AT&T Laboratories 94.5, AT&T Bell Laboratories, 1994.

[16] J. A. Halderman, B. Waters, and E. Felten. A convenient method for securely managing passwords. *Proceedings of the 14th International World Wide Web Conference (WWW 2005)*.

[17] A. Herzberg and A. Gbara. Trustbar: Protecting (even naive) web users from spoofing and phishing attacks. 2004. `http://eprint.iacr.org/2004/155.pdf`.

[18] M. Jakobssen and A. Young. Distributed phishing attacks. 2005. `http://eprint.iacr.org/2005/091.pdf`.

[19] Jefferson Wells Inc. Microsoft Phishing Filter Feature in Internet Explorer 7 and Windows Live Toolbar. 2006. `http://www.jeffersonwells.com/`

client_audit_reports/Microsoft_PF_IE7_IEToolbarFeature_Privacy_Audit_20060728.pdf.

[20] J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure applications of low-entropy keys. *Lecture Notes in Computer Science*, 1396:121-134, 1998.

[21] P. Oorschot and S. Stubblebine. Countering identity theft through digital uniqueness, location cross-checking, and funneling. *Financial Cryptography*, 2005.

[22] Anti-Phishing Working Group. http://www.antiphishing.org.

[23] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. *Proceedings of the 14th Usenix Security Symposium, 2005.*

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

[25] S. Schechter, R. Dhamija, A. Ozment, I. Fischer. The Emperor's New Security Indicators: An evaluation of website authentication and the effect of role playing on usability studies. *IEEE Security & Privacy*, 2007.

[26] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk email. *Learning for Text Categorization*, 1998.

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

[28] M. Wu, R. Miller, and S. L. Garfinkel. Do Security Toolbars Actually Prevent Phishing Attacks. *CHI*, 2006.

[29] M. Wu, R. Miller, and G. Little. Web Wallet: Preventing Phishing Attacks by Revealing User Intentions. *SOUPS*, 2006.

[30] K. Yee and K. Sitaker. Passpet: Convenient Password Management and Phishing Protection. *SOUPS*, 2006.