Password Rescue: A New Approach to Phishing Prevention

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

July 7, 2006

Abstract

A phishing attack exploits both the enormous scale of the web and the fact that users are often enormously confused about what they can trust. Scale allows the phisher to get many responses to his attack, even though the probability of any given user responding is low (it costs the phisher no more to send a million emails than to send one). The enormous confusion about trust allows the phisher make a copy of a bank web-site look as trustworthy to the victim as the original. Previous approaches to this problem have tried to solve the problem by preventing useful information leaking to the phisher; for example by alerting the user to suspicious or low reputation sites. Generally this is done at the client (typically in a browser plugin or add-on).

We propose a scheme that in several respects is a radical departure from previous approaches. First, we make no attempt to prevent information leakage. Rather, we try to detect and then rescue users from the consequences of bad trust decisions. Second, we harness scale against the attacker instead of trying to solve the problem at each client. Thus our scheme increases in efficacy with the scale of deployment: it offers very little protection if a small fraction of users participate, but makes phishing almost impossible as the deployment increases. Finally, we make clear that small trials of our system would prove little. The scale requirements of Password Rescue make it suitable for large deployment or not at all. HotSec seems like the best forum for such ideas.

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 [4] 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 information that they know that are supposed to keep secret. Several approaches to the phishing problem have approached it as a detection problem: they invest a great deal of effort in detecting (*e.g.* in the browser) the fact that a site is phishing. We believe that any algorithm that attempts to *automatically* classify a web-site as phishing (or suspected of phishing) must grapple with the following two constraints:

1. Blocking a connection to a suspected phishing site is unacceptable.
2. Warning users about a suspect site does not get them to change their behavior.

The first point is merely common sense; false positives prevent users from accessing innocent sites. Unless an algorithm can be guaranteed to produce no false positives blocking connections is too risky. The second point is a consequence of the user trust confusion referenced above. The user who lands on a phishing web site already demonstrates confusion about what they can trust. It cannot be assumed that they will notice, understand and act on any pop-ups or warnings delivered by anti-phishing technology running on the client. Recent user studies bear this out: Wu [6] demonstrates that users ignore warnings and pop-ups provided by

anti-phishing toolbars. Dhamija *et al.* [1] confirm that users "proceeded without hesitation when presented with warnings," and that many of the cues designed to help them were ignored or confused them. The second major problem with warning users based on a client-only detection is that, 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.

Our approach is to rescue users from phishers rather than stop information leakage, and to make the scale of a phishing attack work against rather than for the phisher. While it is very difficult to *prevent* users leaking information, it is actually quite simple (as we shall see) to *detect* that leakage has occurred. By relaying the userid of a compromised account to the bank as soon as we detect the leakage we can deprive the phisher of his spoils. If we considered each user alone this would still be a difficult task: the fact of a user typing a password at an unfamiliar site is not actionable. Users share passwords among sites all the time. However, by aggregating the information across many users we can build a far more reliable indication of a phishing attack.

Our argument is provocative for several reasons. First, we make no attempt to prevent information leakage. Expecting users to make good security decisions based on warnings flies in the face of common sense and growing evidence [6, 1]. Second, for the phisher this is a numbers game, and we claim we combat him best by using the scale of the attack against him. Like many security problems there are anti-phishing technologies which have efficacy inversely related to the scale of deployment. That is, a simple anti-phishing toolbar might work well if 1% of people use it (and hence it is not attacked) but not at all if 90% of people do. The approach we present has efficacy that *increases* with the scale of deployment. Finally, our approach, depending as it does on scale, is not amenable to a small trial deployment. Thus we present no data or feedback from actual users. And yet we claim that a scheme such as this probably offers the best hope of protection from a large scale plague. Thus we believe a forum such as HotSec ideal in which to present such an idea.

It is worth mentioning that other innovative approaches have been offered. Oorschot and Stubblebine propose a authentication by means of a hardware personal device [3]. Ross *et al.* [5] ensure that the information that leaks to a phisher is not useful, rather than trying to prevent the leakage.

2 Our Scheme

Our goal is to halt an attack in which a Phisher lures many 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 responsibility of identifying and adding credentials (*uid*, *pwd*) to the protected list; detecting when a user has typed protected credentials into a non-whitelisted site, and reporting that to 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. We will now look at each of these tasks in more detail.

2.1 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.

Identifying credentials to be protected. Essentially, any password, typed at any domain, is added to the protected list. 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. Since it would not be safe to store the credentials in the clear, what we actually store in the protected list is:

$$P_0 = [dom, hash(pwd), hash(uid), t],$$

where $hash$ is a cryptographic hash, and t is the current time. 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).

Detecting when protected credentials are typed. It must be expected however that phishers will employ any means possible to conceal their intent from our defences. An excellent account of several Javascript obfuscating techniques is given in [5]. To handle any and all tricks that phishers might employ we access the keystrokes before they reach the browser scripts. How this is implemented is obviously platform dependent. We use native calls in user32.dll on Microsoft Windows XP that allow our plug-in to get keystroke events directly. Similar calls are available under Linux and the major variations of Unix; the only real requirement is that we can be sure that no scripts running in the browser can confuse the plug-in as to what was typed, or the order in which it was typed. At each key typed, we compute the hashes of possible passwords ending with the last typed character, and check against the appropriate hashes in the protected list. If the hash matches a protected list $hash(pwd)$ value, it means that the just-typed string matches a password on the protected list.

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:

$$C_{report} = [(dom_1, hash(uid_1), t_1), (dom_2, hash(uid_2), t_2), \cdots], dom_R, hash(IP)],$$

where $[(dom_1, hash(uid_1), t_1), (dom_2, hash(uid_2), t_2), \cdots]$ is the vector of domains with the conflicting password and their respective *uid* hashes and last login times, and IP is 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.

2.2 Server Role: aggregation and decision

The server has the role of aggregating information from many users. As detailed in Section 2.1, C_{report} is sent when protected credentials are typed into a non-whitelisted site. This is in itself of course, not proof that an attack is in progress. It means either that the user is knowingly using the same password at a protected and a non-whitelisted site, or that she has been fooled into entering the data by a phishing site. Thus we must distinguish innocent cases of password re-use from phishing attacks. This task is much simplified by the fact that the server aggregates the data across many users. The key for differentiation is that password reuse pattern for a legitimate site will be statistically distributed, while the reuse pattern of a site phishing a particular bank will be highly correlated with that bank. In other words, having a single user typing her BigBank credentials into an unknown site may not be alarming, but having several users type BigBank credentials into *the same* unknown site within a short period of time is definitely alarming.

For our initial evaluation, we assume a site is included in the black list if after receiving five reports for the same non-whitelisted site dom_R, there is at least one "phishable" site in the intersection of the five vectors. This simple rule 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 reuse), aggregated across many users, the server is in a position to identify and stop an attack.

2.3 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.

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

- The attacking domain *dom*
- The hashes hash(*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.

Mitigation. On receiving an attack report from the server dom_R can initiate actions to takedown the attacking site *dom*, and to 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.

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.

3 Analysis and Attacks

3.1 Why Does Scale Matter?

We have mentioned several times that the efficacy of the solution we propose depends on the scale of the deployment. This is so because it is the aggregation of PRU events across many users that allows the server to determine that an attack is in progress. No individual client can make this determination: the fact that a user has typed a password at an unfamiliar site is not actionable on its own. Only the accumulation of those events makes the problem clear.

Deployment to a large number of clients also makes it more attractive for institutions such as BigBank to set up the email account referred to in Section 2.3 and act in a timely manner to do takedown and mitigation.

3.2 Why notify the target and not the victim ?

Observe that even when we conclude a site is phishing, we do not inform the user. This may seem strange or contradictory at first, but there are several reasons for it. Most importantly, *we do not not have a trust channel to the victim*. The findings of [6, 1] indicate that a pop-up window or warning will likely be ignored. We do not have the victim's e-mail, and, even if we did, a "Dear PayPal Customer" e-mail is merely reminiscent of the original phishing email.

The bank (*i.e.* the target) is actually the party most motivated and well positioned to take action. It is in a position to verify whether our information is correct, it is a position to put (possibly temporary) partial blocks on the account, it is in a position to verify the authenticity of the suspect site. And has the financial motivation to limit the losses and to minimize the impact and inconvenience to its customers. And, assuming deployment is large enough, any investments will benefit a significant portion of its customers. In fact, we believe one of the strengths of or method is exactly to put all the mitigation measures in the hands of a large, interested, institution.

3.3 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.

There are several approaches to prevent a client from reporting. The most direct are:

- Flushing the protected list: that is, erase all protected credentials by filling the list with junk entries.
- Hosting on a whitelisted domain: that is, trick a trusted server into serving the content.
- Tricking the user into mistyping the password: that is cause the password re-use check to fail, while giving the attacker enough information to determine the password.

Space doesn't allow a detailed treatment of the ways of preventing these exploits, but each of them has straightforward fixes.

There are also several ways to try to prevent the server from detecting an attack, even though the clients send reports. The principal variations are :

- Distributed attack [2]: the phisher uses many URLs to make blacklisting hard.
- Redirection attack: *i.e.* have a single switch URL that redirects each visitor to a unique site. The phisher needs only as many URLs as victims, but no phishing site ever gets used twice.

To prevent distributed attacks we combine our client reports with traffic information. A site with no traffic history is given a threshold of one in the password re-use detection system. To prevent redirection our client actually reports all redirects within the last 30s, so the switch URL rather than the end phishing site will be caught as the common factor between the victims.

References

[1] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. *CHI*, 2006.

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

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

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

[5] 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.*

[6] M. Wu. Users are not dependable: How to make security indicators to better protect them. *Trustworthy Interfaces for Passwords and Personal Information*, 2005.