Mitigating Man in the Middle Attack over Secure Sockets Layer

Yogesh Joshi[1], Debabrata Das[1], Subir Saha[2]
[1]International Institute of Information Technology Bangalore (IIIT-B),
Electronics City, Bangalore, India.
{yogesh.joshi,ddas}@iiitb.ac.in
[2]Nokia-Siemens Networks, M G Road, Bangalore, India.
subir.saha@nsn.com

Abstract— **Phishing is a social engineering mechanism to steal the user's credentials which are then used for identity theft leading to financial benefit. Currently majority of Phishing attacks are very unsophisticated as they focus on collecting just the credentials and do not try to validate in real time whether the received credentials are correct. It is obvious that next generation Phishing attacks will, in real time, try to check the credentials and also try to exploit the same. It is easy for a Phisher to behave as a man-in-the middle (MITM) between the user and the targeted site which is being phished. The problem with MITM attack is all the heuristics like monitoring domain name for special characters, using blacklists, page analysis etc , fail to restrict the Phisher. One of the significant literature available in this area i.e., PwdHash, which is successful for attacks when the user is on a URL other than genuine website. In this paper, we have proposed and implemented a novel approach to solve MITM over SSL which uses the genuine website URL. To tackle such attacks we propose hashing the user password with the public key of the server's digital certificate. This approach beats the MITM, since the MITM receives the hash of the original password which cannot be reused. We prove our concept with a browser plugin.**

Keywords- Man in the middle, Phishing, Security, Identity Theft,

I. INTRODUCTION

Phishing is the act of stealing user information by pretending to be a trustworthy entity. In the simplest avatar of Phishing attack, a Phisher sets up a bogus website, which closely resembles the targeted original website. It then sends out a bulk of spam e-mails, purporting to be from a legitimate organization, which convinces the user to visit the counterfeit website. Of the many users who get the e-mails, a few fall prey to the attack and give out their credentials. It is well known that Phishers rely on almost any kind of social engineering methods including email, telephone call, people to people communication, SMS, IM. They also leverage other technical subterfuge to lure the victims to the spoofed webpage.

Apart from this mechanism of Phishing attacks, other technical subterfuge schemes plant crimeware onto user's PC which intercepts any information which can be cashed upon by the Phisher such as usernames, passwords, SSN's etc. According the latest reports from Anti-Phishing Working Group (APWG) [7] about 35 percent of the computers were infected with malware.

Yet in another dangerous scenario a Phisher can run its own network node e.g. WiFi access point which allows travelers and even other user to free Internet connectivity. Once user connects to such network node for access to Internet, it is obvious that user's information can be compromised easily.

According to the recent Gartner survey More than 5 million U.S. consumers lost money to phishing attacks in the 12 months ending in September 2008, a 39.8 percent increase over the number of victims a year earlier [1]. In phishing attacks indirect losses are much higher due to the customer service expenses, replacement of account and decreased use of online services in the face of widespread fear about security of online transactions. To fight phishing the strategy must include continuous fraud detection, strong user authentication, and out-of-brand transaction verification for users, a Gartner analyst confirms.

As online crimes got stronger so did the authentication mechanisms. Traditional authentication mechanisms involved sending the user credentials in clear text. This subjected the credentials to variety of attacks, from the hackers. Thus, to provide added security Transport Layer Security/Secure Sockets Layer (TLS/SSL) was introduced. This protocol works by authenticating the server to the client and vice a versa thereby each party confirming the authenticity of the other party. SSL does this by having each party use a digital certificate signed by a trusted third party. However as in most cases current Internet users don't have signed digital certificate; thus most of the websites authenticate only the server to the user. This definitely provides security to the extent that any data that is being eavesdropped on is totally encrypted.

However, using the social engineering method a Phisher can easily plant himself as a Man-in-the-middle (MITM) between a user and the Phished websites irrespective of the sites being secured or not. Once a Phisher is successful in putting itself as MITM, it can seamlessly grab any information encrypted or otherwise information. Figure 1 shows a Phisher in between the user and the original website. It is trivial for such Phisher, now, to act as echoing entity and hence steal any information exchanged between the user and the original site.

978-1-4244-4793-0/09/$25.00 © 2009 IEEE

As most of the website including banking website use the simple SSL or TLS for HTTPS connection, it is clear that the password and userid (or any other credential) exchanged between user and the sites are clearly visible to this MITM Phisher. In essence, currently all known websites using HTTPS, exchange authentication credentials which are not subjected to further protection beyond what HTTPS provides allowing MITM attacks possible even in secured websites.

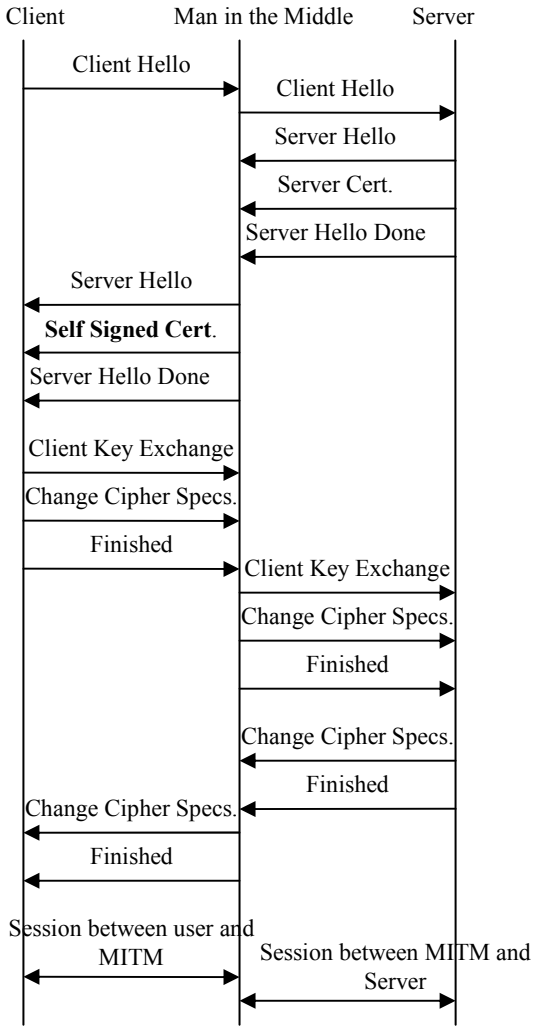

Figure 1: MITM over SSL

A few well referred approaches have been suggested to avoid man-in-the-middle attacks such as Multi-factor authentication [5], digital certificates [6]. While most of these solutions disallow the attackers to use the stolen credential in future times, but these cannot stop attackers in stealing information in the current session in which the attacker has planted itself.

Blake Ross et. al. in a paper proposed PwdHash [2], in which a browser plugin tries to tackle MITM attacks by hashing the user password with domain specific parameters. The advantage of this approach is, even if a Phisher successfully convinces a user to visit the spoofed website, the hashed password that the Phisher receives will be hashed along with the domain of the fake website. Hence this hashed password is expected to be of no value to the attackers as it will not work when used in original website because the original website expects a password to be hashed with its domain parameters.

We believe while [2] provides good protection against majority of phishing attacks of today, it may not be able to stop attackers who are in a position to control the DNS of the victim in some manner. There are at least 2 possible yet known situations in which attackers can exploit DNS to return domain information of attacker's choice,(i) DNS Cache Poisoning and (ii) Running of local DNS.

DNS Cache poisoning consists of changing or adding records in the resolver cache, either on the client or the server, so that a DNS query for a domain returns an IP address for an attacker's domain instead of the intended domain. This can happen through improper software design, misconfiguration of name server, or maliciously designed scenarios exploiting the traditionally open-architecture of the DNS system

The other situation where DNS can be in control of attackers is the cases of so called WiFi Phishing [8]. Let's look at a victim who is desperately looking to use a WiFi in the airport. The attackers runs WiFi network of his own with the same SSID that the victim is expecting to use. As most of the public WiFi do not use any encryption, it is trivial for attackers to remain in between the victims and all the websites that the victim visits. Albeit it is trivial for attackers to run a DNS server of its own which may return what attackers needs for its attacks. Once such a Phisher who can take control of DNS or runs its own DNS, attacks the victim PWDHash does not help at all. We have proposed and implemented a novel idea (explained in detail in Section 2), to over come the above challenges of MITM for Phishing.

The rest of the paper has been structured as follows. Section II provides the proposed solution and implementation. Section III shows a comparative study of our solution and PwdHash. Section IV gives a concrete solution to types of phishing. Finally section V concludes the work.

II. Proposed Solution and Implementation

We thus, propose the hashing of user password with the SSL certificate parameters instead of URL parameters (as done in [2]) using a client side script, say a browser plugin, before submission to the website. The hashing algorithm we use is SHA-1.

In figure 2, we present call flow of our proposed solution to mitigate MITM for phishing. As our solution is using some information embedded in the PKI certificate of the server, there is almost no way the attacker can break the system. For example if the attacker uses the certificate of the genuine server in the first place such that credential is hashed with the parameter of the target server, it will devoid the attacker in the first place to steal the hashed credentials.

As explained below, our solution does have huge benefit compared to PwdHash as we do not need any password

bootstrapping to start with. The rational we use in justifying strength of our system is that the PKI certificate ecosystem is built with strong security. It will be difficult for attackers even in WiFi phishing context to fool the system by running their own root authority.

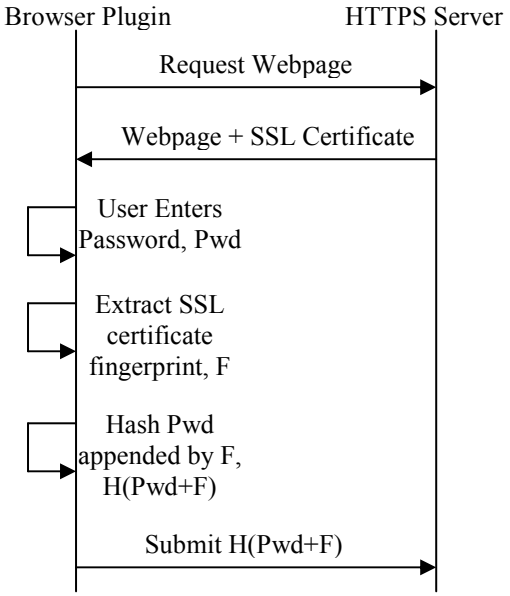

Figure 2: Call flow of proposed solution to mitigate MITM

Unlike PwdHash, we do not depend on user for password change after the plugin is put to use, we would therefore require a change on the server side, which compares the user input password with the hash of the password in the database along with the SSL certificate parameter, instead of direct comparison.

The goal of the plugin is to make sure that the user experience is not changed in anyway. As said in [3], the solution cannot rely on the user for any support in order to avoid getting phished. For the same reason, we decided to go with the server side changes, since they would be one time. The authors again stress that this solution targets those HTTPS attacks where the URL remains genuine, but the Phisher replaces' the server's digital certificate with his own certificate.

Design Consideration:

Client Side

We develop Firefox plugin to demonstrate the proposed solution. When the browser requests for an HTTPS website, the digital certificate of the website is sent to the browser. The plugin then extracts the fingerprint of the digital certificate using the Javascript support provided by Mozilla for Firefox development. Post the extraction of the fingerprint of the certificate the plugin then looks for out for the password fields in the web page. These input fields are identified in HTML with the type parameter set to password, <input type="password">. Once the user keys in the "password" in the password field and submits the form, the plugin appends the extracted fingerprint to the "password" in the password box and hashes the combination, and replaces the contents of the password box with this new hash, thus submitting the hashed password. The straight forward way to implement the hashing in a browser is using the readily available SHA-1 implementation.

Server Side

The compensation to be paid for the user experience continuing to be same is that we have to make changes at the server side. Without this scheme the server would normally accept the user password and compare it with the password stored in the database. The change required in this approach is instead of comparing the user entered password with that in the database, the server would have to generate a hash of the user password in the database appended by the digital certificate fingerprint of the SSL certificate and compare this with the plugin generated password as explained above.

Attack Analysis on Plugin

Here we discuss some of the possible attacks on our solution and also provide analysis why our solution will survives such attacks.

Our approach can be defeated using the well known Javascript techniques like mock password fields, asynchronous submitting of user input and so on. To get around with such Javascript attacks, the plugin employs intelligent detection of password fields. The aforementioned Javascript attacks can be done in three ways:

1) **Mock password field using password field**: In this attack the Phisher is indeed using the input type as "password", however for every key press he is also appending the user's password to a hidden field too. The Javascript for the same is as follows:

```
<form>
<input type="hidden" name="rogue" value="">
<input type="password" name="fair" onKeyPress="
this.form.rogue.value+=String.fromCharCode(event.keyCode); ">
</form>
```

2) **Asynchronous submission**: In an asynchronous submission attack the Phisher embeds an asynchronous Javascript in the web page, which submits the characters to the Phishers web site as the user types each character. This kind of an attack foils the attempt to hash the password just before submission, because before the user clicks the login, the password is already submitted. This can be done using a simple Asynchronous Javascript and XML (AJAX) snippet.

In either of the approach, the user is typing in a password box. Thus, to detect such a behavior we propose a scheme where the Javascript is disabled as soon as the user is typing in the password box. As soon as the user looses focus of the password textbox the Javascript is again enabled. This approach again supports our notion of no involvement of the user in the hashing process.

3) **Mock password field using text field**: In this attack the Phisher doesn't use a password field as an input field, but a simple text field. For every key press of the user the Phisher stores the key stroke in a hidden field and displays an asterisk in the text field, making it appear as a password box.

```
<form>
<input type="hidden" name="rogue" value="">
<input type="password" name="fair" onKeyPress="
this.form.rogue.value+=String.fromCharCode(event.keyCode);event.keyCode=42; ">
</form>
```

One solution to the above attack is to invoke a listener for every key press. The listener would then check the textbox in which the data is being entered for the presence of an asterisk. If the asterisk is present the Javascript is disabled and the defense mechanism proceeds as mentioned in previous subsection.

III. Comparative Study: PwdHash and Our Proposed Solution to Mitigate MITM

In this section we do a comparative study of PwdHash of [2] and our approach and enlist the pros and cons of each approach. To proceed with this section the authors assume the reader has knowledge about working of PwdHash. The PwdHash suggests to tackle Phishing attacks using hashing of the credentials by the domain name parameters of the target server, on the other hand we in this paper talk about attacks over SSL such that the URL and the website presented to the user is a legitimate one, however the data being sent and received is over a channel such that it no longer remains safe from the eavesdropper. Though one approach is for non SSL and the other one is for SSL both the approaches have the same goal of defending against man in the middle attacks. We thus feel that by a comparative study of both these approaches we can come up with a single solution which can tackle all types of Phishing attacks.

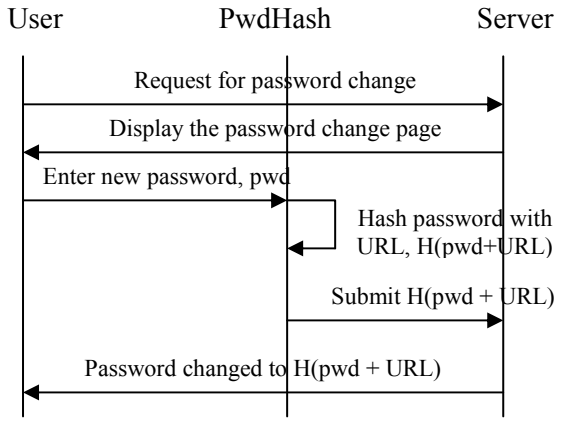

Figure 3: Password Change, post installation of PwdHash

This essentially changes the user experience since it incurs an overhead to manually change the password. As we can see in Figure 3, the user has to request for password change on the website, and enter the new password, on being prompted with password change page. PwdHash then hashes the entered password with the URL of the server and this hash is submitted to the server. Thus, the password stored on the server is the hash of the user entered password and the URL. This process has to be repeated for all the websites.

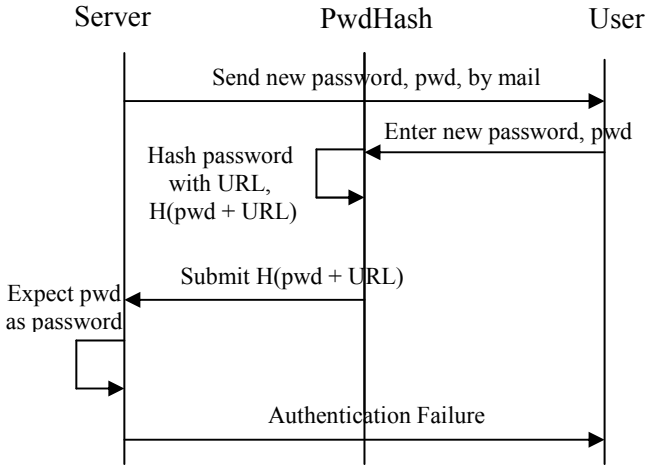

Figure 4: Authentication failure for new password sent via E-mail to the user in case of PwdHash

Adding to this if a user forgets his password and requests for a password change it also causes an inconvenience to the user. After a request for a new password the server mails the user, a new password. Now as can be seen from Figure 4 the user's browser has the plugin installed which hashes the password of the user with the domain name. Also for the server to authenticate the user, the hashed password has to be present at the server side, which is done by manually changing the password as explained in Figure 3. Thus, when the user, enters the new password the plugin hashes this and the server receives the hashed password, however at the server side since the password hasn't yet been manually changed the authentication fails.

Apart from these obvious limitations of PwdHash, it also has another restriction for the site owners. As the password is hashed along with the URL of the sites, it is difficult to change the URL as the stored hashed value will not match now. We must remember that websites store hashed password and not raw (clear text) password to protect from insider theft. On the contrary PKI certificate does not change as frequently as we expect the URL of a bank to change. This makes our solution easier for website providers.

Due to the above overhead and complexity in certain use cases, in our approach we resort to server side changes. For the same scenario our approach would submit the hash of the new password to the server, and due to the server side changes, the server calculates the hash of the new password, sent to the user and compares it with the received hash, if they match the user is granted access.

Since PwdHash relies on the current domain name for the salt, in attacks like ARP spoofing where in spite of the user entering a genuine URL in the address bar the user is redirected to the malicious website, the domain name used for salting will be the genuine one and the resulting hashed password can be used by the Phisher on the original website. On the other hand the approach we suggest employs hashing using the SSL certificate parameters, such as the fingerprint of the certificate or the public key, thus, an MITM who replaces the original certificate with a self signed certificate cannot reuse the hashed credentials.

Lastly [2] talks about allowing users who do not have the plugin installed, avail the hashing of password functionality by providing a webpage that generates the hashed passwords. This idea opens up a new window to the Phisher, since the Phisher can create a replica of this webpage and capture the clear text passwords. These clear text passwords can then be used on the original website

IV. Complete Solution For Phishing

Building on the above observations, we can say one particular solution is not sufficient for the phishing attack. We thus, suggest using multiple heuristics for detection and mitigation of phishing attack. Most of the solutions use blacklisting approach as the source for detection of phishing sites, however there is always a time between the site being hosted and it being reported as phishing site. In this section we propose a concrete solution which mitigates the attack without depending on a blacklist or any other third party.

In [4] we have talked about the random credential submission on a website, which seems to be simple but is concrete. This approach of ours, will definitely force the Phisher to carry out his attack in real time, since after the reporting of loss of credentials, the user might change his credentials. Moreover, to circumvent this solution the Phisher will be forced to be MITM. We leverage on this situation and put together the solution proposed in this paper with that of PhishGuard [4] to detect and mitigate phishing conclusively. Such a combination would detect phishing websites, that give a fixed response, and if the phishing site is an MITM then it would hash the password with the SSL certificate and submit it. In the first case we can successfully detect the phishing site, in the second case however we mitigate the Phishing attack.

The work done by PwdHash is also useful to attacks that don't use SSL certificates but are MITM. However, it changes the user experience to a certain extent.

Conclusion

In this paper we have proposed a novel approach to tackle man in the middle attack over secure connection (SSL), without any change to the user experience. The idea has been backed up by a browser extension. The paper also discusses the various attacks that can take place on the plugin. Lastly we do a comparative study of our approach with PwdHash. We also refer our previous work and provide a complete solution for Phishing.

Acknowledgment

The authors thank Motorola Research India and IIIT-Bangalore for supporting this research work. One of the authors, Subir Saha, used to serve at Motorola Research India before joining Nokia Siemens Networks.

References

[1] http://www.gartner.com/it/page.jsp?id=936913

[2] *Blake Ross*, *Collin Jackson, Nick Miyake, Dan Boneh, John C Mitchell,* "Stronger Password Authentication Using Browser Extension", Proceedings of the 14th Usenix Security Symposium, 2005.

[3] Min Wu, Robert C. Miller, Simson L. Garfinkel, "Do Security Toolbars Actually Prevent Phishing Attacks?", Conference on Human Factors in Computing Systems, 2006.

[4] Yogesh Joshi, Samir Saklikar, Debabrata Das, Subir Saha, "PhishGuard: A Browser Plug-in for protection from Phishing", IEEE COMSOC sponsored 2nd International Conference on Internet Multimedia Services Architecture and Application (IMSAA-08), 10-12th December, 2008, Bangalore, India.

[5] http://en.wikipedia.org/wiki/Two-factor_authentication

[6] http://www.ietf.org/rfc/rfc3280.txt

[7] http:///www.antiphishing.org/reports/apwg_report_H2_2008.pdf

[8] htttp://www.wlanbook.com/wifi-phishing/