

# Let's Sink The Phishermen's Boat!

Teo Sze Siong

[teo.sze.siong@f-secure.com](mailto:teo.sze.siong@f-secure.com)

**Abstract:** In recent years, the boom of e-commerce has changed the way people do business and manage their money via online banking. This technology wave has driven banks to invest huge amount of dollars in security infrastructure to protect their daily business operations and to increase their customers' confidence towards them. However, the paradigm has now shifted to the client side attacks as most users are unguarded and vulnerable against cyber attacks which are launched either through technical perspective or social engineering means. Many users still do not understand the risk that they face even when they are using their own trusted computers to perform online banking protected with 2-factor authentication security.

In this paper, an advanced form of phishing attack will be discussed to show the risk how criminals might steal the entire fund from an online banking account protected with daily transaction limit and bypassing the 2-factor authentication system. This type of attack is able to work in stealthy mode without showing any theft symptoms in the bank account balance to keep the victims in the dark. Challenges and limitations encountered by the existing phishing detection techniques will be also identified and reviewed to understand the applicability of each technique in different scenarios.

As a step taken to combat phishing attacks, the concept of 'website appearance signature' will be presented and explained how this concept can be applied to detect unknown phishing websites. This has been a great challenge in the past since most phishing website detection tools verify the reputation of a website using a database of blacklisted URLs. In addition, a Proof-Of-Concept application employing the 'website appearance signature' combining with conventional phishing detection techniques will be demonstrated to see its' accuracy and effectiveness as a phishing website detection tool.

## **1. Introduction**

According to Gartner report, United States adult lost about USD3.2bil in year 2007 due to phishing frauds. Banking industry spent millions of dollars to deploy security systems such as the 2-factor authentication system as a step to increase their customers' confidence. However, there is a lack of public awareness regarding the risk when performing online banking without observing the proper security measures.

In this paper, I present techniques that might be used by criminals to trick their victims into revealing their banking information without raising suspicious symptoms to maintain continuous access. Besides, some techniques that might be employed by malware to steal sensitive information are also described in this paper to show the risk when performing financial related activities on a malware infected machine.

## **2. Related techniques**

In recent years, we have seen different techniques being used by malware to help criminals stealing information and remain stealthy either through social engineering or technical implementation. Some of those techniques can be observed from the following:

### **2.1 Hosts file modification (Pharming attack)**

In this technique, malware will modify the '**hosts**' file of the operating system by adding an entry to make the legitimate banking website's hostname to resolve to the attackers web server IP address. When the victim enters the URL of a legitimate website, the web browser will load the fake banking website hosted by the attacker. This type of attack is known as '**pharming**'. This technique will probably fail when the malware is not running with sufficient privilege to modify the hosts file. Besides, this technique may also trigger an alert on systems installed with IDS/IPS that monitors or prevents hosts file changes. The hosts file for Windows platform by default is located at **C:\Windows\System32\drivers\etc\hosts** while the hosts file for UNIX based platform is usually located at **/etc/hosts**. On Windows system, the DNS cache can be cleared to reload the hosts file by issuing a command like '**ipconfig /flushdns**' while on UNIX based system is usually done by restarting the DNS cache daemon that is selected to be used.

## **2.2 Keystroke monitoring**

Very often, malware such as Trojan horses are designed to listen on keystrokes to steal information such as credit card numbers, usernames and passwords on infected machines. Nevertheless, this method is getting less effective as the use of one time password or security token authentication in 2-factor authentication of online banking system makes the stolen information become worthless. This challenges the attackers to shift their strategy to much more sophisticated form of attacks.

## **2.3 Fake windows form**

This is basically a form of social engineering attack that shows a professional looking window form that looks like an interface of legitimate software. These fake graphical user interfaces usually ask for credit card number or financial account information such as PayPal to complete purchase or registration of a particular product. Although this method is less seen nowadays, but still there are victims who fall into this kind of trap.

## **2.4 Web browser modification**

Web browser functionalities are usually modified by malware through DLL injection or installation of malicious plugins to steal sensitive information entered by the user or stored on the local machine. These attacks usually target Microsoft Internet Explorer web browser due to its insecure design that allows dangerous code execution in ActiveX components.

## **2.5 API hooking (user mode or kernel mode)**

API hooking technique has become increasingly popular in recently years employed by malware to prevent antivirus software detection. In the past, API hooking at user mode such as IAT/EAT patching and inline function hook were commonly used for reverse engineering purposes or modify the behavior of a legacy application without source code. The paradigm has now shifted to kernel land using techniques such as Direct Kernel Object Manipulation (DKOM) or patching the System Service Descriptor Table (SSDT). The most dangerous part of this technique when misused is that it can be used not just to steal information, but also capable of hiding the malware process, network sockets, registry keys and files to avoid detection. These features that make it stealthy are commonly known as rootkit behaviour. Fortunately, tools such as F-Secure BlackLight, Rootkit Revealer or SDT Restore are able to detect the presence of rootkit.

### Advanced phishing attack

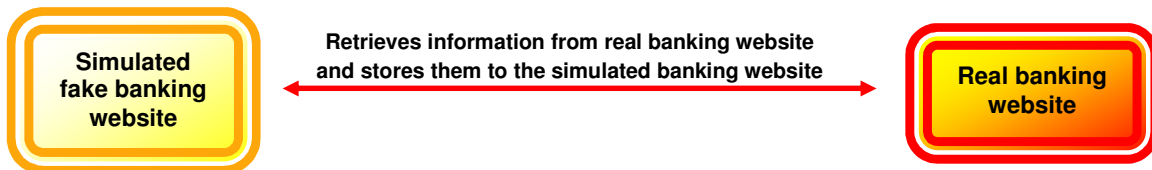
This section describes an advanced form of phishing attack that might be employed by criminals to steal the entire fund from an online banking account protected with daily transaction limit and bypassing the 2-factor authentication system. The risk of such attack is that criminals can transfer out all the money from their victims' bank account in several transactions while keeping them in the dark. Although there are several attack techniques can overcome the 2-factor authentication security exists presently, but those techniques work in a **'hit-and-run'** way thus not capable of drawing out the entire fund from an online banking account that is protected with daily transaction limit.

The approach for this attack is to remain stealth by showing the victims fake information such as last login date/time, transaction history, balance amount, etc. that should reflect in their real banking account to prevent the victims from knowing that their banking account is under attack. Therefore, the attacker can have ample time to transfer all the money in several transaction days. Below shows the flow of such attack in different steps:

2.6 Victim logs in to the fake banking website using their username, password and one-time-use security token generated from security device or smartcard provided by bank. The attacker uses the login information entered by victim at the fake banking website to login to the real banking website.



2.7 The attacker retrieves information such as account number, last login, transaction history, etc. from the real banking website and stores them to the simulated fake banking website database.



2.8 In online banking systems protected with 2-factor authentication, a security token is required from the user for each transaction to be performed. Whenever the victim enters a security token to perform transaction, the attacker uses the security token entered at the fake website to perform fund transfer from the victim's banking account to their money mule's account.



- 2.9 Since the security token will expire within a short time frame, automating the attack in real-time is important to ensure successful fund transfers. The attacker can easily automate a web browser to perform login and transactions by sending mouse clicks and keystrokes using functions exported by user32.dll such as SendInput(), PostMessage(), SendMessage(), mouse\_event() or keybd\_event(). This method will be a lot simpler and less effort to implement than simulating a web browser with SSL support to automate the attack.
- 2.10 Communication with banking websites must usually go through encrypted channel (HTTPS) thus intercepting the data received from web server at socket level is not a good choice. To retrieve decrypted information in web browser received from the server, the attacker can just create a browser plugin such as Browser Helper Object to inspect the information via Document Object Model (DOM) of the loaded page. In addition, automation of information retrieval can also be done from web browser user interface by sending mouse clicks and virtual keys such as CTRL+A and CTRL+C to copy the selected information to clipboard. Then, the information in the clipboard can be reformatted and stored into the simulated fake banking website to fool the victim.
- 2.11 If the victim's banking account is preset with daily transaction limit, then the attacker will have to perform several transactions in different days. In this case, the attacker needs to reduce the use of security tokens by avoiding redundant login attempts. This can be done by making the web browser reload the page automatically every minute to prevent from session expiry. Step 1.3 to 1.6 is repeated until the balance in victim's banking account is emptied.

**Reasons of automating the attack through GUI at server side:**

1. It is a lot simpler than hooking HttpRequestA() in wininet.dll therefore there is a high probability that more criminals might use this technique
2. Hooking wininet.dll doesn't work for web browsers are implemented using API calls directly to Winsock APIs and proprietary or open SSL libraries
3. Compared to API hooking or using Browser Helper Object (Man-In-The-Browser attack) at client side, simulating the web request at server side does not trigger any IDS/IPS software that detects hooking behavior or browser integrity tampering
4. Storing the malicious logic code at server side for automation also makes security analyst hard to find out what was done and what is being done in the attack when performing forensic analysis on victim's computer

**Phishing Website Detection using '*website appearance signature*'**

Phishing is a form of social engineering attack to gain the trust of their victim in revealing sensitive information therefore it is technically hard to detect fraudulent website 100% accurately. In this section, the concept of '*website appearance signature*' is introduced to assist the detection of similar websites and to show how it can be applied to detect phishing websites.

First, the screenshot of a rendered website is captured in 24-bits color depth. Since two similar images will contain the similar color palettes and similar amount of pixels in the same group of palettes, thus the color mean values for red, green and blue value of an image can be used as a signature to identify their similarity. In this approach, we only need to know the amounts of similar color pixels so the arrangement or orders of color pixels are ignored. This is due to some legitimate websites' content might be aligned to the center while the similar content on phishing websites are aligned to the left or even right.

There are 8 bits in a byte. Each pixel in a 24-bits color depth image is represented by 3 bytes. The color of a pixel is combined from red, green and blue therefore the value for each element can range from 0 to 255. To obtain the signature of a rendered website screenshot, I've chosen a simple way by using the mean values for the red, green and blue element of an image.

Below shows example of four same size images followed by their red, green and blue mean values. The first image is the original screenshot of the rendered PayPal website while the second image is a messed up image modified from the original screenshot by cutting the image into multiple pieces. Third image is the fake version of PayPal website with the contrast and brightness level slightly tweaked to make sure the color values are different. The last image shows a totally different image which is the screenshot of rendered 2Checkout.com website.

1. paypal.bmp – A screenshot of the real PayPal website



[Mean values]

Red: 226.26349166666665, Green: 232.64016333333333, Blue: 236.67534166666667

2. messed.bmp – A messed up image modified from paypal.bmp



[Mean values]

Red: 226.26936333333333, Green: 232.64310833333334, Blue: 236.67663166666668

- 3. fake.bmp – A screenshot of fake PayPal website [contrast and brightness level tweaked]



[Mean values]

Red: 225.603835, Green: 231.98625166666667, Blue: 236.01825500000001

- 4. 2checkout.bmp – A screenshot of the real 2Checkout.com website



[Mean values]

Red: 207.40960000000001, Green: 220.19798166666666, Blue: 213.34901500000001

2.12 **Image similarity detection**

Based on the pattern of these data, it is clear that when two images containing similar color palettes and similar amount of pixels in the same group of palettes, the mean values for red, green and blue are almost identical or near to each other. Thus, the RGB mean values of an image can be used as a signature to check for similarity of rendered web appearance. The following formula can be used to calculate the similarity percentage between two images:

**r1 – Red color mean value of image-1,                      r2 – Red color mean value of image-2**  
**g1 – Green color mean value of image-1,                    g2 – Green color mean value of image-2**  
**b1 – Blue color mean value of image-1,                    b2 – Blue color mean value of image-2**

$$rDiff = |((r1 - r2) / 256)|, \quad gDiff = |((g1 - g2) / 256)|, \quad bDiff = |((b1 - b2) / 256)|$$

Therefore,

$$100 - ((rDiff + gDiff + bDiff) * 100) = \% \text{ of similarity}$$

Example calculation:

**Difference of paypal.bmp and messed.bmp**

$$rDiff = |((226.26349166666665 - 226.26936333333333) / 256)| = 0.00002293619791671875$$
$$gDiff = |((232.64016333333333 - 232.64310833333334) / 256)| = 0.0000115039062500390625$$
$$bDiff = |((236.67534166666667 - 236.67663166666668) / 256)| = 0.0000050390625000390625$$
$$100 - (0.000039479166666796875 * 100) = \underline{\underline{99.9960520833333203125 \% \text{ similar}}}$$

**Difference of paypal.bmp and fake.bmp**

$$rDiff = |((226.26349166666665 - 225.603835) / 256)| = 0.0025767838541666015625$$
$$gDiff = |((232.64016333333333 - 231.98625166666667) / 256)| = 0.002554342447916640625$$
$$bDiff = |((236.67534166666667 - 236.01825500000001) / 256)| = 0.002566744791666640625$$
$$100 - (0.0076978710937498828125 * 100) = \underline{\underline{99.23021289062501171875 \% \text{ similar}}}$$

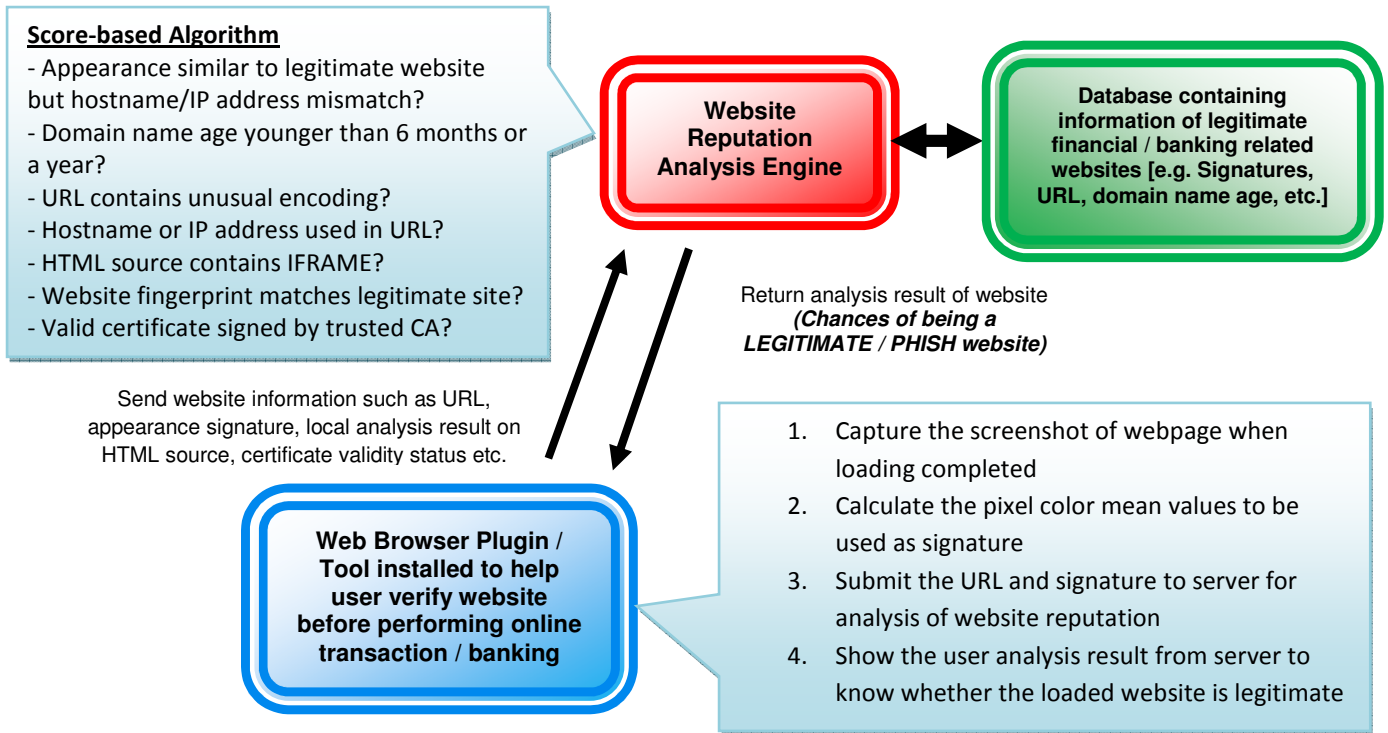
**Difference of paypal.bmp and 2checkout.bmp**

$$rDiff = |((226.26349166666665 - 207.40960000000001) / 256)| = 0.0736480143229165625$$
$$gDiff = |((232.64016333333333 - 220.19798166666666) / 256)| = 0.0486022721354166796875$$
$$bDiff = |((236.67534166666667 - 213.34901500000001) / 256)| = 0.091118463541666640625$$
$$100 - (0.2133687499999998828125 * 100) = \underline{\underline{78.66312500000001171875 \% \text{ similar}}}$$

This approach is far more effective than creating a signature based on the layout structure of HTML or Javascript source of a website because some phishing websites employ obfuscation technique or showing a similar appearance entirely using Flash content.

2.13 Application of website appearance signature

Using the earlier described technique to obtain website appearance signature and percentage of similarity between two websites, the algorithm can be applied to create a simple anti-phishing system. Since it is much easier to obtain information of legitimate websites than blacklisting phishing websites, we can make use of that information in an example system like below:



Example of a basic anti-phishing system

## References

### Websites:

- [1] <http://en.wikipedia.org/wiki/Pharming>
- [2] <http://www.gartner.com/it/page.jsp?id=565125>
- [3] [http://en.wikipedia.org/wiki/Color\\_histogram](http://en.wikipedia.org/wiki/Color_histogram)
- [4] <http://en.wikipedia.org/wiki/Histogram>
- [5] [http://en.wikipedia.org/wiki/Man-in-the-middle\\_attack](http://en.wikipedia.org/wiki/Man-in-the-middle_attack)
- [6] [http://www.apacs.org.uk/media\\_centre/press/03.10.07.html](http://www.apacs.org.uk/media_centre/press/03.10.07.html)
- [7] [http://en.wikipedia.org/wiki/Root\\_mean\\_square](http://en.wikipedia.org/wiki/Root_mean_square)
- [8] <http://en.wikipedia.org/wiki/Mean>
- [9] [http://en.wikipedia.org/wiki/Money\\_mule](http://en.wikipedia.org/wiki/Money_mule)
- [10] <http://www.seclab.tuwien.ac.at/papers/antiphishdom.pdf>
- [11] <http://www.cs.berkeley.edu/~asimma/294-fall06/projects/reports/cordero.pdf>
- [12] <http://www.pythonware.com/library/pil/handbook/image.htm>
- [13] <http://iplab.naist.jp/member/daisu-mi/miyamoto-jwis2007.pdf>
- [14] <http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf>
- [15] <http://www.codeproject.com/KB/system/hooksys.aspx>
- [16] [http://www.f-secure.com/weblog/archives/VB2007\\_PresentationSlides.pdf](http://www.f-secure.com/weblog/archives/VB2007_PresentationSlides.pdf)
- [17] [http://www.f-secure.com/weblog/archives/VB2007\\_TheTrojanMoneySpinner.pdf](http://www.f-secure.com/weblog/archives/VB2007_TheTrojanMoneySpinner.pdf)
- [18] [http://www.owasp.org/index.php/Man-in-the-browser\\_attack](http://www.owasp.org/index.php/Man-in-the-browser_attack)
- [19] [http://www.owasp.org/index.php/Session\\_hijacking\\_attack](http://www.owasp.org/index.php/Session_hijacking_attack)
- [20] [http://en.wikipedia.org/wiki/Browser\\_Helper\\_Object](http://en.wikipedia.org/wiki/Browser_Helper_Object)
- [21] [http://www.theregister.co.uk/2006/07/13/2-factor\\_phishing\\_attack/](http://www.theregister.co.uk/2006/07/13/2-factor_phishing_attack/)
- [22] [http://www.theregister.co.uk/2007/04/19/phishing\\_evades\\_two\\_factor\\_authentication/](http://www.theregister.co.uk/2007/04/19/phishing_evades_two_factor_authentication/)
- [23] <http://msdn2.microsoft.com/en-us/library/bb250489.aspx>
- [24] [http://www.planb-security.net/wp/503167-001\\_PhishingDetectionandPrevention.pdf](http://www.planb-security.net/wp/503167-001_PhishingDetectionandPrevention.pdf)
- [25] [http://www.f-secure.com/v-descs/mimail\\_s.shtml](http://www.f-secure.com/v-descs/mimail_s.shtml)
- [26] [http://www.symantec.com/security\\_response/writeup.jsp?docid=2007-040208-5335-99&tabid=2](http://www.symantec.com/security_response/writeup.jsp?docid=2007-040208-5335-99&tabid=2)
- [27] [http://vil.nai.com/vil/content/v\\_126303.htm](http://vil.nai.com/vil/content/v_126303.htm)
- [28] [http://www.symantec.com/security\\_response/writeup.jsp?docid=2004-101116-3507-99&tabid=2](http://www.symantec.com/security_response/writeup.jsp?docid=2004-101116-3507-99&tabid=2)
- [29] [http://www.f-secure.com/v-descs/trojan-spy\\_w32\\_zbot\\_hs.shtml](http://www.f-secure.com/v-descs/trojan-spy_w32_zbot_hs.shtml)