Trust on Web Browser: Attack vs. Defense

Tie-Yan Li and Yongdong Wu

Infocomm Security Department
Institute for Infocomm Research (I^2R)
21 Heng Mui Keng Terrace, Singapore 119613
{litieyan, wydong}@i2r.a-star.edu.sg

Abstract. This paper proposes a browser spoofing attack which can break the weakest link from the server to user, i.e., man-computer-interface, and hence defeat the whole security system of Internet transaction. In this attack, when a client is misled to an attacker's site, or an attacker hijacks a connection, a set of malicious HTML files are downloaded to the client's machine. The files are used to create a spoofed browser including a faked window with malicious event processing methods. The bogus window, having the same appearance as the original one, shows the "good" web content with "bad" activities behind such as disclosing password stealthily. Once the attack is mounted, even a scrupulous user will trust the browser that is fully controlled by the attacker. We further propose several countermeasures against the attack.

1 Introduction

With the rapid development of Internet and web technologies, most of the online applications are built on or assisted by WWW. As shown in Figure 1, a typical web-based transaction consists of several components linked together for serving requests. Being requested, the web server receives a piece of reply data from application database. The web server then sends the data passing through the firewall, routers towards the requesting machine. Finally, the data is shown on the screen-user interface. We denote the secure link from the server to the user as *trust path*. Along the long link, various malicious attacks can be launched. For instance, the web site may suffer from database attack, web attack, and DDoS attack; the network may be assaulted by attacks such as eavesdropping, session hijacking, and routing disruption; virus may also disclose the information in the user's machine.

Because the important data may be eavesdropped, replayed or modified, security is the most important concern as of online transaction. The web site has a lot of security policies and means to guard the transaction system. Firewalls protect the hosts from malicious outside attacks; Intrusion Detection Systems monitor the networks and hosts all the time; Secure channels are built with security protocols (e.g., SSL [3]); And the host is well protected from virus. After all the defense are in place, are the shown data at the recipient side trustworthy? Most of the users will say "Yes". Why? The browsers produced by big players

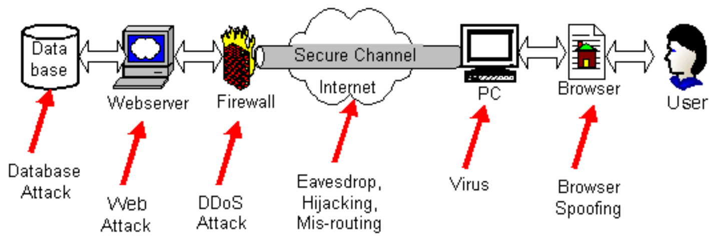


Fig. 1. Trust path

Microsoft or Navigator show the security sign on the screen. Regretfully, the answer is “No” because the security signs used for man-machine-interface can be easily reproduced with “Web spoofing” technique ([4] [5] [6] [7]). Therefore, the whole trust path can be easily broken at this weakest point by web spoofing. This topic is also mentioned in [2], which established several principles for designing secure user interface.

In this paper, we propose a browser spoofing attack simply using applet and frame to demonstrate that the popular commercial browsers (Microsoft Internet Explorer and Netscape Navigator) are not trustworthy. Comparing to web spoofing, browser spoofing can simulate dynamic event response as well as static visual appearance. To this end, the attacker lures the client to accept a faked HTML page at first. This HTML file is used to create a new window with a method `window.open()`. The bogus window, having the same appearance as the original browser window, shows the web content of the target server. By appending Java applet methods, the bogus browser can respond to the client’s input events without being suspected. This attack allows an adversary to observe and modify all web pages sent to the client’s machine, and observes all information input by the client. This attack is effective even though the web sever is SSL-enabled. Further, a user is unable to detect the attack even if he checks the security features, such as the security lock. In sum, the bogus browser, including the bogus window and the methods, is almost the same as the genuine browser from the viewpoint of the user. Meanwhile, the countermeasures against this attack are also provided.

The organization of the paper is as follows. Section 2 analyzes relevant works. Section 3 introduces the generic attack scheme. Section 4 addresses our implementation. Section 5 proposes the countermeasures toward the attack. At last, we conclude the paper and point out the future directions.

2 Related Works

Of all security techniques against Internet attacks, SSL3.0 [3] is the *de facto* standard for end-to-end security and widely applied to do secure transactions such as Internet banking. In the ISO/OSI network reference model, SSL is located in the transport layer. Thus, SSL provides message confidentiality and integrity in transport layer. The higher application layer uses SSL as the secure transport channel. However, SSL only authenticate the transport connection, it does not authenticate the content sent to the recipient. That is to say, SSL guarantees that the received message is authentic and confidential in the transmission, but it does not care about the message before or after transmission. In all, SSL protects the connection rather than the applications. In a web application, a security lock is normally shown at the bottom of a browser window if SSL is completed successfully. Users trust this sign of security and further on, conduct “secure transactions”. Therefore, if an attacker manages to produce a security lock appearing in the right place of the user’s screen, he will convince the user to trust a faked site.

Felten et al. [4] proposed a web-spoofing attack. In the scheme, an attacker stays between the client and the target site such that all web pages destined to the user’s machine are routed toward the attacker’s server. The attacker rewrites the web pages in such a way that the appearances of these pages do not change at all. On the client browser, the normal status and menu information bar are replaced by identical-looking components supplied by the attacker. The attack [4] can prevent HTML source examination by using JavaScript to hide the browser’s menu bar, replacing it with a menu bar that looks just like the original one. To attack a SSL-enabled web server, the attacker sends the client a certificate of an innocent person to avoid punishment. Although the method is quite straightforward, it is hard to launch because the attacker has to obtain the corresponding private key of the innocent person. It is also easy to be detected since a cautious user can also detect this attack by checking the security properties of the server.

Lefranc and Naccache [5] described malicious applets that use Java’s sophisticated graphic features to rectify the browser’s padlock area and cover the address bar with a false https domain name. Mounting this attack is much simpler than [4] as it only demands the insertion of an applet in the attacker’s web page. The attack was successfully tested on Netscape’s Navigator. But the attack is not successful when it is tested on Microsoft’s Internet Explorer because a warning message is added in the end of the popup window. To overcome this shortcoming, the authors suggested to patch an artificial image. However, a weird image may also alert the client that an attack is under way. Moreover, Horton et al [6] and Paoli et al [7] adopted the patch method.

Ye et al [11] [12] proposed a trusted path solution to defend against web spoofing. The authors set up the boundary of the browser window with different colors according to certain rules. They defined an internal reference window whose color is randomly changed. Any malicious web content will fail to control a browser window due to uncontrollable inset/outset attributes. Therefore, if a new pop-up window has a different color from that of the reference window, the user concludes that a web-spoof attack is under way. For the device of small screen (such as hand-held device), this countermeasure is impractical because it is inconvenient to open two windows and switch between the windows. Moreover, the attacker can create a bogus reference window to overlap the original reference window so as to break the defense.

The countermeasures are not limited in these “inside, software based” references. Some approaches [13] [14] used “outside, hardware based” references as the security evidences or trusted devices. Burnside et al [13] deployed a trustworthy camera which takes authentic pictures used for comparison with those on a large screen. Based on visual cryptography techniques, Tuyls et al [14] used an additional “decryption display” as the reference for protect two-way communication across insecure devices. One former approach [8] indicated several design issues of using mobile user devices in legally significant applications.

3 Attack on Trust Path

Trust relationship can be established in many ways. Authentication is one of the most popular means in reality. It checks what the claimant knows (e.g password), what the claimant has (e.g., security token), or what the claimant is (e.g. fingerprint). In the application of Internet transaction based on SSL protocol, the browser shows a security sign (e.g. security lock) if the server knows the private key conforming to a certificate. The browser vendors (e.g., Microsoft) declare that the server site is authentic if the lock sign is lit. However, the security lock-like sign is not enough to be used to trust a server due to our attack addressed in the following.

3.1 Attack Model

In this section, we design a model to attack SSL-enabled web application (because a non-secure web server is easy to be spoofed [4]). Refer to Figure 2, four kinds of participants are involved in this model: user, client, server and attacker. The normal transaction process is as follows. The user visits a web server via a web browser (client) such as Netscape Navigator or Microsoft Internet Explorer (IE). The SSL-enabled server owns a certificate issued by a certificate authority. When a user requests a secure page, SSL protocol authenticates the server and generates a session key for the secure communication between the client and the server. Meanwhile, the security lock is lit in the client’s browser status line if the server is authenticated. Additionally, if the user clicks the security lock, the security information such as server certificate information will be shown on the data area of the popup window.

To mount the browser spoofing attack, the attacker should lure the client to accept malicious packets so as to sit in the middle between the target server and client. To this end, the attacker starts the attack in three ways before forging a SSL session.

1. The attacker controls a router or proxy in the communication path. The request toward a secure web server will be re-routed to an attacker’s server.

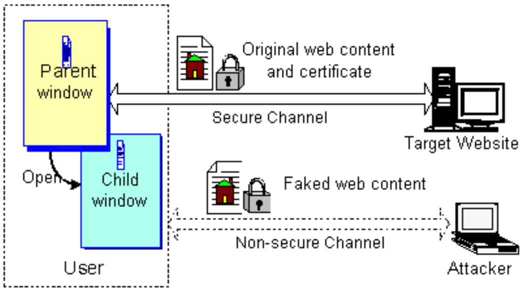


Fig. 2. Attack model

2. The attacker can hijack the communication session between the client and the server. When an initial SSL protocol request message is intercepted, a faked web page is sent to the client via non-SSL channel instead of the original one.
3. The client could be lured to browse the attacker's site pretended to be a trusted server. This method is often used since it can be done easily.

Thus, the attacker can insert, delete and tamper the communication data. S/he may feed the browser with the faked web content and a manipulated certificate. In sum, a malicious server is set up to communicate with the client "securely".

3.2 Spoofing Method

After forcing the client to receive malicious packets, the attacker sends a HTML file to the client so as to create a spoofed browser. This malicious HTML page may

1. create a new window with the method `window.open(attackerURL, "BogusWindow", "menubar=0, scrollbars=0, directories=0, resizable=0, toolbar=0, location=0, status=0")`, the parameter "0" indicates that the attacker disables the default browser window configuration.
2. draw a bogus status line and other GUI components with a security lock.
3. display the content of the target page and
4. create event response functions. For example, when the user checks the security property, an artificial dialog should be popped up to convince the user that all the security information is correct.

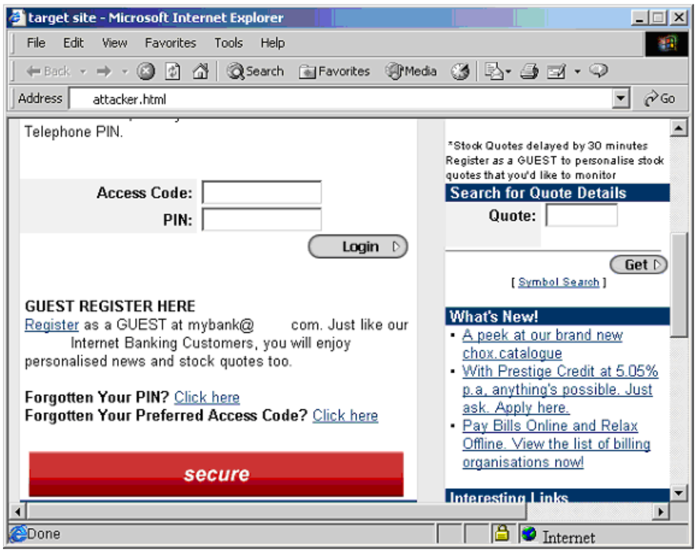
After creating the new window, there are two windows on the screen of the user's machine. One is original and the other is bogus. The spoofed one is in the front and is enlarged to overlap the original one. Although the spoofed window has the same interface as the genuine one, it is actually under the control of the attacker.

4 Implementation

Currently, two popular browsers are Microsoft Internet Explorer and Netscape Navigator. Our attack can be mounted on either of them. Because the implementations for both browsers are similar, we illustrate the attack to Internet banking between a SSL-enabled target site (Internet bank) and a user using Microsoft IE as a client. The target server has a certificate issued by some Certificate Authority (e.g., verisign.com) whose public key is embedded in Internet Explorer.

4.1 Spoofing the Browser Window

Most of the Internet surfers are familiar with the browser interface of Microsoft Internet Explorer, which looks like Figure 3. The user interface is a main window. In this window, its title bar includes a title and three system buttons. The following bar includes the menu bar and some shortcut icons. The address bar indicates the current web site, followed by the page content. In the last line, the status information is shown. The status information comprises of the icon of Internet Explorer logo, a security lock (a closed lock icon) representing that the present communication is secure, and some others.


target site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Login
Get
secure
Done
Internet

attacker.html

Telephone PIN.

Access Code:
PIN:

GUEST REGISTER HERE
Register as a GUEST at mybank@ com. Just like our Internet Banking Customers, you will enjoy personalised news and stock quotes too.

Forgotten Your PIN? Click here
Forgotten Your Preferred Access Code? Click here

*Stock Quotes delayed by 30 minutes
Register as a GUEST to personalise stock quotes that you'd like to monitor

Search for Quote Details
Quote:

[Symbol Search]

What's New!
- A peek at our brand new chox.catalogue
- With Prestige Credit at 5.05% p.a. anything's possible. Just ask. Apply here.
- Pay Bills Online and Relax Offline. View the list of billing organisations now!

Interesting Links

Fig. 3. Spoofed browser

Figure 3 is a logon page of an Internet banking system. It requires the bank customer to input a pair of account and password for identity verification. Here, SSL protocol is executed and a security lock is shown on the status line. All the personal customer information are attempted to be encrypted with a session key and uploaded to the bank server. Unfortunately, the above interface is bogus and the personal information is sent to the attacker! The faked interface is created with our browser spoofing attack.

Technically, this bogus interface is generated after the client accepts a malicious HTML file (see Table 1) which creates a new window without status line. The new window is split into two frames. One is named as *upperFrame*. The *upperFrame* showing the web page `attacker.html` of the target site, is provided by the attacker. But from the view of the client, the content is not abnormal because the content is correct. The other frame is forged to be the status line of the genuine browser, and is split further into 5 sub-frames. Those sub-frames display IE icon with the HTML file `ielogo.html`, an earth icon with the HTML file `earth.html` and a security lock (closed) icon with the file `lock.html`.

Table 1. Frame settings

```
<FRAMESET ROWS="*,18" frameborder="YES" border="1"
                                framespacing="0">
    <FRAME NAME="upperFrame" SRC="attacker.html" >
    <FRAMESET cols="*,24,24,24,150" frameborder="YES" border="1"
                                framespacing="0">
        <frame name="ielogo" scrolling="NO" MARGINHEIGHT="0"
                                noresize src="ielogo.html">
        <frame name="status" scrolling="NO"MARGINHEIGHT="0"
                                noresize src="clearbg.html">
        <frame name="progress" scrolling="NO" MARGINHEIGHT="0"
                                noresize src="clearbg.html">
        <frame name="lock" scrolling="NO" MARGINHEIGHT="0"
                                noresize src="lock.html">
        <frame name="earth" scrolling="NO" MARGINHEIGHT="0"
                               noresize src="earth.html">
    </FRAMESET>
</FRAMESET>
```

To cheat the careful users, `lock.html` provides the security lock icon as well as its actions. Table 2 is the code of `lock.html` where `showLayer()` processes the click event. When the user clicks on the security lock icon, the response action will display the certificate information of the bank server as the genuine browser does.

4.2 Spoofing the Browser Methods

To enable the user to read the source of a web page, Microsoft IE provides "Viewing the Document Source" menu-item in the menu. In the web-spoofing attack proposed by Felten et al. [4], if the user chooses from the spoofed menu bar,

Table 2. Display security lock

```
<html>
        <head>
            <title>Untitled Document lock</title>
            <meta http-equiv="Content-Type" content="text/html;
                                  charset=iso-8859-1">
        </head>
        <body bgcolor="#c8d0d4">
            <A onclick= "window.top.showLayer()"> <IMG src="lock.bmp"> </A>
        </body>
</html>
```

the attacker would display the original HTML source instead of the faked code. The attacker can also discourage the user from choosing the browser's "view document information" menu-item. Those attackers skilled in web development can further substitute the URL address field with a text field easily to forge the target server address. Thus, we do not implement the bogus menu-items, URL and the related methods. We focus on the security lock icon and the related mouse events. In a genuine browser, if a user clicks on the security lock so as to confirm the server further, the server information will be displayed on a movable popup window which is able to accept the user mouse input. To simulate the above functionalities for cheating the cautious user, the attacker creates the HTML tag <LAYER> other than the popup window to display the certificate information to avoid a warning message, while the response methods on mouse events are implemented in an applet.

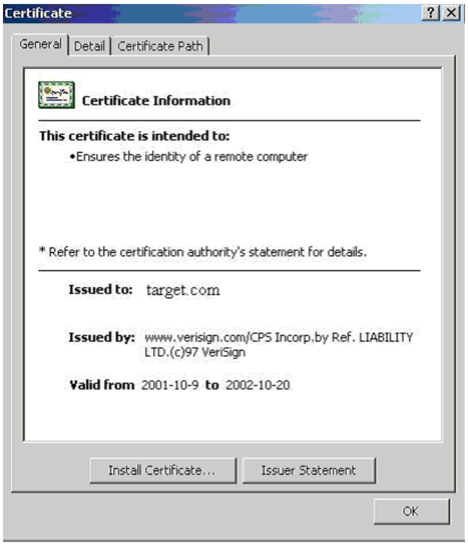
Spoofing the Certificate Window. As mentioned in [9] [10], a HTML tag <LAYER> is a frame that can be absolutely positioned. It can occupy the same 2D spaces as another frame. A layer looks like a frame with a document property that is in turn an object, with all the properties of the top-level document object. It captures events in the same way as the top-level window or document. The basic properties are the same as the other HTML elements. The layer-specific attributes are "`top`", "`visibility`" and "`id`" properties: "`top`" property specifies its position so as to move the layer; "`visibility`" controls whether the layer is displayed;"`id`" identifies the layer. Table 3 sets up the layer to forge the certificate window. Table 3 is the code to generate the layer whose `id` is "*Layer1*", default visibility attribute is "*hidden*", and size is 400×500 pixels. Its top-left coordinate is (80, 40). The document loaded in this layer is the applet encapsulated in a jar file "`Spoof.jar`". This applet is enabled to communicate with the HTML JavaScript functions by setting the attribute "MAYSCRIPT=true". The code in Table 3 is appended to the server web, as well as the jar file "`Spoof.jar`".

As the normal browser, the applet shows the initial page of the certificate. As shown in Figure 4, the page includes 3 tabs: General, Detail and Certificate Path. The different tab page can be switched freely when the intended tab is clicked. Because the contents of these tabs are well known in advance, the attacker

Table 3. Embedding the spoof applet

```
<div id="Layer1" style="position:absolute; visibility:hidden; width:400px;
                         height:500px; z-index:1;left: 80px; top: 40px">
      <applet code=SpoofApplet archive="Spoof.jar" width=410 height=477
                         MAYSCRIPT=true>
      </applet>
</div>
```

encapsulates them in a jar file in advance so that the client can not detect the attack by checking the certificate, the expiry and any other information.



Certificate Information

This certificate is intended to:

•Ensures the identity of a remote computer

* Refer to the certification authority's statement for details.

Issued to: target.com

Issued by: www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign

Valid from 2001-10-9 to 2002-10-20

Fig. 4. Certificate window

Spoofing the Event Methods. The above step creates a static bogus certificate window which can cheat average users. To cheat careful users, the malicious applet should response to the user input dynamically. To this end, the bogus browser processes 5 kinds of events.

1. Click on the closed lock icon: The code in `lock.html` (see Table 2) indicates that the function `showLayer()` will make the layer visible when the *onClick* event occurs. The code in module `showLayer()` in Table 4 finds the document at first, then finds the layer based on layer `id` value "*Layer1*" attribute

which is defined in Table 3. The event response module changes the visibility attribute of the layer to be "*visible*" from "*hidden*". That is to say, the certificate window will be shown when the user clicks on the lock icon. In order to save time for next display of certificate information, the layer is hidden other than destroyed when closing certificate window.

Table 4. Change visibility of a layer

```
<Javascript>
Function showLayer()
        {var doc=window.top.frames[0].document;
         var layerstyle=doc.all["Layer1"]["style"];
         if( layerstyle.visibility=="hidden") layerstyle.visibility="visible";
         else layerstyle.visibility ="hidden"; }
</Javascript>
```

2. Click on the tab button: the corresponding tab page will be exposed. Class `JTabbedPane` provides an easy way to switch between the tabs.
3. Moving the certificate window: when the position of mouse is obtained, the new position of the layer can be set. However, the layer can move in the area of the bogus window only. This weakness may help a cautious user to detect the attack. Unfortunately, few users check the security by moving a window.
4. Click on the internal buttons: When a user clicks the buttons whose actions are restrained to the applet itself, the action procedures are easy to be realized. The exemplary button is the "Install Certificate ..." in the General tab page.
5. Click on external buttons: The buttons including the "OK" button and the "close" button in the top-right corner can close the certificate window. Because the button click event is received by the applet, the applet should pass it to the HTML/JavaScript page. By searching the `JSObject` tree, the method `actionListener` of `Okbutton` calls the Javascript function `showLayer()` in the HTML page. Table 5 illustrates this code.

Table 5. Response to click on lock

```
OkButton.addActionListener(new ActionListener() {
        Public void actionPerformed(ActionEvent e) {
                JSObject win=JSObject.getWindow(theApplet);
                        // theApplet is the name of the malicious applet.
                Object[] args=new Object[0];
                win.call("showLayer",args);}
});
```

6. When the user clicks on the system close button on the top-right corner of the browser window to close the window, the attribute of layer visibility is set to be "*hidden*".

5 Countermeasures toward Browser Spoofing

Our spoofing attack is very concise and effective. Any script kiddie can launch it easily. However, it is quite difficult to defend it. The reasons are

– Most users are not security expert. Their using behaviors are hard to be changed. If the users are used to trust a security lock, they will not be alerted by other anomaly events, e.g. changes on status bar.
– Even the expert themselves may be cheated by the faked lock and the faked certificates. Worst, if the terminal holding the browser is totally controlled by malicious environment, all tries of tracing a packet, redirect the connection to prove the original site would receive manipulated replies.

Actually, the failure mainly results from no clear visual sign to help the users' making a correct decision (i.e. a security lock is apparently not enough for application level security). To counter this attack, we have to rely on more visible and clear sign indicating certain trustworthy event. Two countermeasures are described in the following.

5.1 Disabling Configuration of Status Bar

In the browser spoofing attack, a new window is created with faked status line which indicates a bogus site to be authentic. To thwart this attack, the browser should disable status bar configuration when a new window is created. This modification is easy to implement without affecting the functionality of browsers. Thus, a security lock in status bar always shows the trustworthy site if and only if the server certificate is authentic. However, this countermeasure requires the user to pay attention to the status of the small security lock.

5.2 Synchronizing Colors

It is easier for a user to trust the web content if the colors are synchronized within a window. For example, dynamically controlling the color of a designated part of a window based on certain security rules. It is very hard for an attacker to synchronize the color without passing the security rules. Refer to Figure 5, the main menu bar, whose color is randomly set up by internal functions, is labelled as a standard reference. The boundary of the new generated windows will also show the same colors after accepting a certificate. If the colors are not identical, the new window can not be trusted.

Compared with [12], this solution requires no additional reference window. It could be ideal since it satisfies the necessary requirements in [12]: inclusiveness,

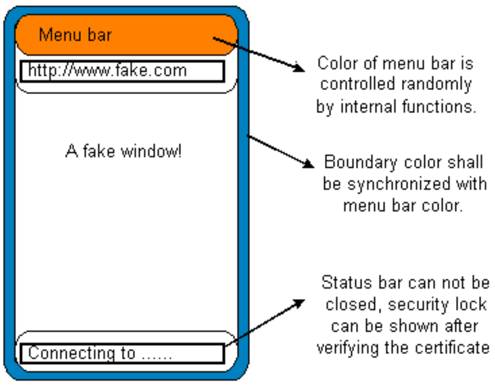

Menu bar
http://www.fake.com
A fake window!
Connecting to ......
Color of menu bar is controlled randomly by internal functions.
Boundary color shall be synchronized with menu bar color.
Status bar can not be closed, security lock can be shown after verifying the certificate

Fig. 5. Color synchronization

Effectiveness, no user work, no intrusiveness. It is also easy to distinguish the bad behavior windows. We believe that the browser spoofing attack is not incurable as long as the browser software providers patch the security hole as the above countermeasures.

6 Conclusions and Future Directions

In this paper, we demonstrated an effective attack - "browser spoofing" that make the browser un-trustable. We introduced a generic attacking scheme and elaborated its implementation. While using browser spoofing is not novel, the visual effect of this experiment can really cheat quite a lot of users. It shows that the trust path from user to web browser is weak, although security protocols like SSL are secure enough for end-to-end security. But the client side "end" security ends merely at transport layer. The gap still exists between the user and its browser, in which languages (i.e. Java, JavaScript) and dynamic properties (i.e. form functions, frames) provide rich effects, yet dangerous.

Although the attack is quite effective, it can be avoided by disabling the "status bar" and carefully detecting any anomaly happened there. In fact, to trust on the web browser, systematical defense technologies will be integrated together. The more complicated the strategies, the more user involvement. The less possible the attackers' following up, the more trustable the content. Hence, the challenge is how to balance the tradeoff between trust and ease of use.

Further on, we will study several secure schemes against the attacks for protecting web browser. We believe that the proposed attack is not incurable with effective trust scheme by a non-cautious user.

References

1. http://www.mymontage.com/.
2. Ka Ping Yee, User Interface Design for Secure System. ICICS, LNCS 2513, (2002)278–290
3. A. Freier, P. Kariton, P. Kocher, The SSL Protocol: Version 3.0.Netscape communications, Inc., (1996)
4. Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web spoofing: An Internet Con Game. 20th National Information Systems Security Conference,(1997) http://www.cs.princeton.edu/sip/pub/spoofing.html
5. Serge Lefranc and David Naccache, "Cut-&-Paste Attacks with Java". 5th International Conference on Information Security and Cryptology (ICISC 2002), LNCS 2587, pp.1–15, 2003.
6. Jeffrey Horton and Jennifer Seberry, Covert Distributed Computing Using Java Through Web Spoofing. ACISP (1998)48–57, http://www.uow.edu.au/ jennie/WEB/JavaDistComp.ps.
7. F. De Paoli, A.L. DosSantos and R.A. Kemmerer, Vulnerability of "Secure" Web Browsers. Proceedings of the National Information Systems Security Conference (1997)
8. Andreas Pftizmann, Birgit Pfitzmann, Matthias Schunter and Michael Waidner, Trusting Mobile User Devices and Security Modules, IEEE Computer, 30/2, Feb, 1997, p. 61–68.
9. Rick Darnell et al, Dynamic HTML. ISBN 0-57521-353-2(1998)
10. Gabriel Torok, Jeffrey Payne and Matt Weifeld, Javascript Primer Plus. ISBN 1-57169-041-7(1996)
11. Yougu Yuan, Eileen Zishuang Ye. Sean Smith, Web Spoofing. (2001) http://www.cs.dartmouth.edu/reports/abstracts/TR2001-409/
12. Eileen Zishuang Ye, Sean Smith, Trusted Paths for Browsers. 11th USENIX Security Symposium, (2002)
13. M. Burnside, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten van Dijk, Srinivas Devadas, and Ronald Rivest The untrusted computer problem and camera-based authentication. 1st International Conference on Pervasive Computing, LNCS 2414,(2002) 114–124.
14. Pim Tuyls, Tom Kevenaar, Geert-Jan Schrijen, Toine Staring, Marten van Dijk, Visual Crypto Displays enabling Secure Communications, Proceeding of First International Conference on Security in Pervasive Computing(2003)12–14