2002 International Computer Symposium

Workshop on Cryptology and Information Security

Title: Browser Spoofing

Abstract: This paper presents an attack which spoofs a browser in the client's machine. When a client visits the attacker's site, or the attacker hijacks a connection, a malicious HTML file will be downloaded to the client's machine. This file instructs to create a spoofing browser including a faked window and event processing methods. The bogus window, having the same appearance as the original browser window, shows the web content of the target site. But the bogus methods provide malicious activities besides the genuine one. From the viewpoint of the client, the browser is still trustworthy although it is at the fully control of the attacker. This spoofed browser can cheat the client even a "secure" connection is built. We design a simple system to demonstrate the attack.

Keywords: Web Security, Dynamic HTML

Authors: Yongdong Wu, Ma Di and Xu Chang Sheng

Laboratories for Information Technology

21, Heng Mui Keng Terrace, Singapore, 119613

{wydong, madi,xucs}@lit.a-star.edu.sg

Browser Spoofing

Abstract: This paper presents an attack which spoofs a browser in the client's machine. When a client visits the attacker's site, or the attacker hijacks a connection, a malicious HTML file will be downloaded to the client's machine. This file instructs to create a spoofing browser including a faked window and event processing methods. The bogus window, having the same appearance as the original browser window, shows the web content of the target site. But the bogus methods provide malicious activities besides the genuine one. From the viewpoint of the client, the browser is still trustworthy although it is at the fully control of the attacker. This spoofed browser can cheat the client even a "secure" connection is built. We design a simple system to demonstrate the attack.

1. Introduction

With the rapid development of the Internet, the network security becomes more and more important. SSL (Secure Socket Layer)[5] protocol is widely used in the secure transactions such as Internet Banking business. It authenticates the web server and sets up a secure channel for the transaction between the server and client.

Felten [1] *et al*. proposed a web spoofing attacker which is the man-in-the-middle attack. In this attack scheme, the attacker sits between the client and the target site, all web pages destined for the client's machine are routed through the attacker's server. The attacker rewrote the web pages in such a way that the appearances of these pages does not change at all. In the client side, the normal status and menu information are replaced by identical-looking components supplied by the attacker. The attack [1] 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. To attack the SSL-enabled web server, the attacker breaks into a server and sends the client the certificate of the innocent person to avoid punishment. However, it is hard to launch because the attacker has to obtain the corresponding private key. A cautious client can also detect this attack if he checks the security property.

Lefranc and Naccache [2] describes 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 scenario is much simpler than [1] as it only demands the inclusion 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 the strange image may prompt the client that an attack is under way.

In this paper, we propose a browser spoofing attack that the browser is not trustworthy. At first, the attacker lures the client to accept a faked HTML page. 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 site. By appending Java Applets 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 observe all information inputted by the client. This attack is effective even though the web sever is SSL-enabled. Further, it is hard for a client to acknowledge the attack when he checks the security parameters, such as the security lock. In all, the bogus browser, including the bogus window and the methods, is almost the same as the genuine browser from the viewpoint of the client.

Section 2 elaborates the attack. Its implementation is addressed in section 3. Section 4 proposed the countermeasures.

2. Attack scheme

2.1 Model

There are 3 kinds of participants: client, server and attacker. The client visits a web server via his/her own web browser such as Netscape Navigator or Microsoft Internet Explorer (IE). Because a non-secure web server is easy to be spoofed [1], we focus on the SSL-enabled server only, which has a certificate issued by an authority. When a client requests a secure page, SSL protocol provides a mechanism to authenticate the server and generate a session key for the secure communication between the client and the server. Meanwhile, the padlock is lit in the client's browser status line if the web site is authenticated. If the client clicks the padlock, the security information such as server certificate information will be shown.

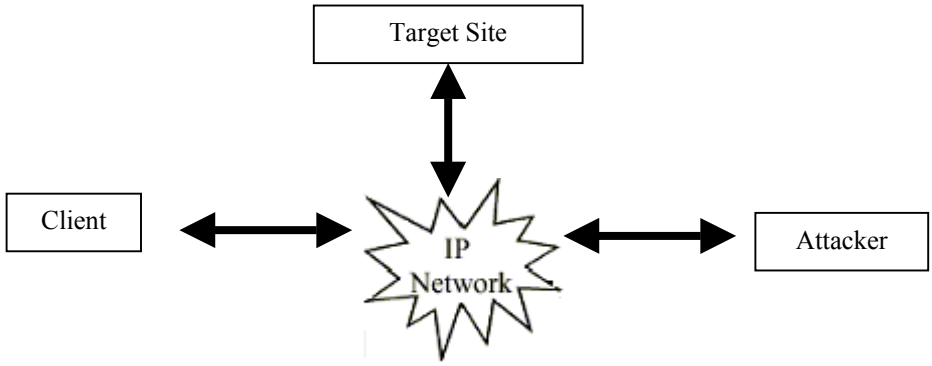

Figure 1: Model

2.2 Attack mounting

To mount the attack, the attacker should lure the client to accept his packets. Generally, the attacker can mount this attack in 3 ways.

(1). The attacker controls a router or proxy in the communication path.

(2). The attacker can hijack the communication between the client and the server.

(3). The client is lured to browse the attacker's site before a trusted server.

If the attacker sits in the middle, he can insert, delete and tamper the communication data between the server and client.

2.3 Browser spoofing

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

- creates a new window with the method *window.open*(attackerURL, "BogusWindow", "menubar=0, scrollbars=0, resizable=0, toolbar=0, location=0, directories=0, status=0"), the parameter 0 indicates that the attacker disables the default browser window configuration and will set a fake one.
- draws a bogus status line with a closed padlock.
- displays the content of the target page
- creates event response functions. For example, when the client check the security property, an artificial dialog should pop up to convince the client that all the security information is correct.

After creating the new window, there are two windows on the screen of the client 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 under the control of the attacker.

3. Implementation

At present, the two popular browsers are Microsoft Internet Explorer and Netscape navigator. Our attack can be mounted on any of them. Because the implementations for both browsers are similar, we demonstrate the attack to Microsoft Internet Explorer only. Here, we discuss the target site has a certificate issued by one CA (Certificate Authority) whose public key is embedded in Internet Explorer. When one visits this target site, an SSL-enabled channel is set up and a closed lock is shown on the status line. If the client clicks on the lock icon, the certificate information window will be popped up. This certificate window describes the issuer, the owner and the signature information. In the following steps, we demonstrate that the above solution is still insecure by spoofing the browser.

3.1 Spoofing the Browser Window

Most of the Internet surfers are familiar with the client interface of Microsoft Internet Explorer, which looks like figure 2. The user interface is a window. In this window, its title bar includes a title and 3 system buttons. The following bar includes the menu bar and some shortcut icons. The address bar indicates the current web site, followed by its content. In the last line, the status information is shown. The status information comprises of the icon of Internet Explorer logo, an important lock icon representing that the present communication is secure, and some others.

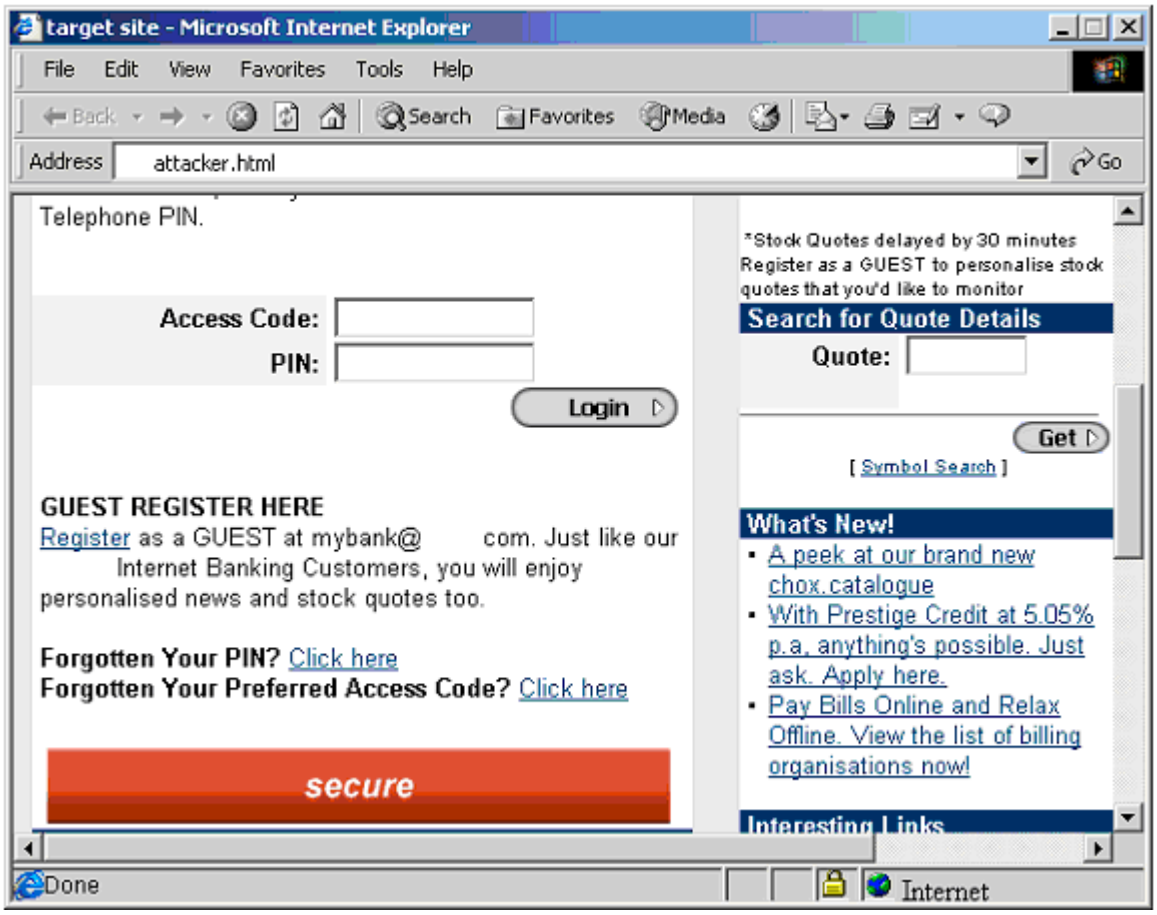

Figure 2: Spoofing browser

Figure 2 is a logon page of an Internet banking system. It requires the bank customer to input his/her account and password pair for identification verification. Here, SSL protocol is executed and a closed lock is shown on the status line. All the personal information is 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. This bogus interface is generated when the client accepts a malicious HTML file which creates a new window without status line. The new window is split into two

frames. One is named as *upperframe*. The *upperframe* shows the web page *attacker.html* of the target site, which comes from the target site and tampered 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 as the status line of the genuine browser. It is 18 pixels in height, and is split further into 5 sub-frames. Those sub-frames display IE icon in the HTML file *ielogo.html*, an earth icon in the HTML file *earth.html* and a closed lock icon in the file *lock.html*. The code to generate the interface is listed in table 1.

Table 1

```
<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 client, *lock.html* includes the closed lock icon, as well as its action when the client clicks on this icon to check the information of the bank server. Table 2 is the code of *lock.html* where *showLayer*() processes the click event.

Table 2:lock.html

```
<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>
```

3.2 Spoofing the Browser Methods

Felten et al. [1] addressed their attack can prevent “Viewing the Document Source” by using JavaScript to hide the browser’s menu bar, replacing it with a menu bar that looks just like the original. If the client chose “view document source” from the spoofed menu bar, the attacker would open a new window to display the original (non-rewritten) HTML source. It can also discourage the victim from choosing the browser’s “view document information” menu item. Thus, we do not implement the menubar and the related methods. The address field can be replaced with a label field, a text field and an icon.

Those skilled in web development can implement the address field easily. Therefore, we focus on the closed lock icon and the mouse events.
The HTML page displays the closed lock icon. When the client clicks on it, the server information window will expose. This window is movable and accepts the client mouse events. We select the HTML tag <LAYER> other than the popup window to display the certificate information because the later is appended with a warning message which will alarm the client. We implement the response methods to the mouse event in an applet.

3.2.1 Spoofing the certificate Window

As mentioned in [6,7], an HTML tag <LAYER> is a frame that can be absolutely positioned and can exist in 2.5 dimensions — that is, it can occupy the same 2D spaces as another frame. It 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 thr other HTML elements. The only layer-specific attributes needed are the *top*, *visibility* and *id* properties. *id* identifies the layer. *top* property specifies its position. Changing *top* property can move the layer. *visibility* controls whether the layer is displayed. Table 3 sets up the layer to forge the certificate dialog. Table 3 is the code to generate the layer whose *id* is "*Layer1*", default *visibility* attribute is hidden, size is 400×500. Its top-left coordinate is (80, 40). The document loaded in this layer is the applet encapsulated in a jar file "*Spoof.jar*", which includes other resource such as icons. This applet is enabled to communicate with the HTML javascript function 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*".

Table 3

```
<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>
```

As the normal browser, the applet shows the initial page of the certificate. Because the content of these tabs are fixed, the attacker can hardcode them in the jar file in advance. Figure 3 illustrates the certificate window, which has the same function as the genuine one: General tab page, Detail tab page and certificate path tab page. The client can check the certificate, read the expiry and any other information. The different tab page can be switched freely when the intended tab is clicked.

Figure 3: Certificate window

3.2.2 Spoofing the Event Methods

The above step creates a bogus certificate window which can cheat a careless client. However, if the client clicks the closed lock icon, there is no certificate information shown on the screen. The malicious applet is used to display the information. It also provides the detail actions on the mouse events from the client. There are 5 kinds of events which may be processed by the bogus browser.

(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 find the layer based on layer *id* value "Layer1" attribute which is defined in table 3. It changes the *visibility* attribute of the layer to be "visible" from "hidden". That is to say, the certificate window will be shown when the client clicks on the lock icon. In order to save time, the layer is hidden other than destroy when a system close click is activated. In case that the client clicks on the lock again, the certificate window can be shown immediately.

Table 4

```
<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

Because the Layer has the *position* attribute, 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 window.

(4) Click on the internal buttons.

When the client clicks the buttons whose actions are restrained to the applet itself, the action procedures are easy to be realised. The exemplary button is the "Install Certificate…" in the General tab page.

(5) Click on external buttons

These buttons, including the "OK" button and the close button in the top-right corner, close the certificate window. Because the button click event is received by the applet, the applet should pass it on 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

```
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);
    }
});
```

Similarly, when the client 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".

4. Counter measurement

In order to mount this browser spoofing attack, a new window without status line is created. Thus, the original browser should disable the capability of status bar configuration when a new window is created.

References

1. Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web spoofing: An Internet Con Game. 20th National Information Systems Security Conference, Maryland, October 1997. http://www.cs.princeton.edu/sip/pub/spoofing.html
2. Serge Lefranc and David Naccache, Cut and Paste Attacks with Java, http://eprint.iacr.org/2002/010.ps
3. Jeffrey Horton and Jennifer Seberry, Covert Distributed Computing Using Java Through Web Spoofing. Australasian Conference on Information Security and Privacy, http://www.uow.edu.au/~jennie/WEB/JavaDistComp.ps.
4. Volker Roth and Vania Conan, Encrypting Java Archives and its Application to Mobile Agent Security. http://www.semoa.org/docs/papers/vroth00d_amec.pdf
5. Alan O. Freier, Philip Karlton and Paul C. Kocher, The SSL Protocol, Version 3.0, http://wp.netscape.com/eng/ssl3/draft302.txt, 1996
6. Rick Darnell et al, Dynamic HTML, 1998, ISBN 0-57521-353-2
7. Gabriel Torok, Jeffrey Payne and Matt Weifeld, Javascript Primer Plus, 1996, ISBN 1-57169-041-7