SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks **00**: 1–13 (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/sec.0000

SSL-Enabled Trusted Communication: Spoofing and Protecting the Non-Cautious Users

Fang Qi[1], Zhe Tang[1], Guojun Wang[1,2,*] and Jie Wu[2]
[1] *School of Information Science and Engineering, Central South University, Changsha, 410083, P. R. China*
[2] *Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA*

Summary

The anti-spoofing community has been intensively proposing new methods for defending against new web-spoofing techniques. In this paper, we analyze the problems within current anti-spoofing mechanisms, and propose a new SSL protected trust model. Then, we describe the attacks on SSL protected trusted communication. In this paper, we also propose the new Automatic Detecting Security Indicator scheme (ADSI) to defend against spoofing attacks on SSL protected web servers. In a secure transaction, ADSI will randomly choose a picture and embed it into the current web browser at a random place. This can be triggered by any security relevant event that has occurred on the browser, and then automatic checking will be performed on the current active security status. When a mismatch of embedded pictures is detected, an alarm goes off to alert the users. Since an adversary is hard to replace or mimic the randomly embedded picture, the web-spoofing attack can not be mounted easily. In comparison with existing schemes, (1) the proposed scheme has the weakest security assumption, and places a very low burden on the user by automating the process of detection and recognition of web-spoofing for SSL-enabled trusted communication; (2) it has little intrusiveness on the browser; and (3) it can be implemented in a trusted PC at an Internet Cafe. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS: Automatic Detecting; Security Indicator; Web-spoofing; Secure Socket Layer; Trusted Communication

1. Introduction

With rapid development of the Internet and web technologies, most of the online secure applications (e.g., on-line banking) are protected with secure socket layer (SSL) protocol [1]. Although SSL guarantees that the received message is authentic and confidential in the transmission, it does not authenticate the interface between users and machines. Therefore, the user may be cheated since the human computer interface (HCI) is not trustworthy, despite an SSL protocol is performed. This HCI attack is also referred to as web-spoofing in web applications.

Web-spoofing often fakes a new bogus window and its methods. The bogus window, having the same appearance as the original browser window, shows the web content of the attacker. In addition, 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 user's machine, and observes all input information of the client.

Here is an illustrative example of web-spoofing, a user likes to visit a bank for online transaction

*Correspondence to: School of Information Science and Engineering, Central South University, Changsha 410083, China. E-mail:csgjwang@mail.csu.edu.cn and bank account management, by clicking a hyperlink. Unfortunately, the user is misled to a bogus web site which creates a spoofed password dialogue windows. According to present web technologies, SSL-protected windows have all the security features, and hence careful users may check the indicators in an SSL-protected web page: a closed lock icon in the status bar, a text of the actual domain name in the status bar, and a closed lock icon in the address bar. The lock icon indicates that the page the user is viewing was delivered to the user securely. Due to the crafty spoofing technology, the spoofed window looks exactly like a real window, and has a bogus status line and other graphical user interface (GUI) components with a security lock. Even a user that takes time to inspect a web site certificate may be cheated since the shown certificate may be bogus. Therefore, the indicators do not guarantee that the received page is authentic. Consequentially, the personal data entered into password dialog boxes will be sent to the attacker not the genuine server.

Web-spoofing attacks have been known for years [2]- [8] and has resulted in business and e-commerce transactions hijacking. To prevent web-spoofing, many solutions [9]-[15] have been proposed. These solutions create obvious logos such as text, symbols, synchronized random dynamic boundaries (SRDs) of the browser [9], a trusted credential area (TCA) [11] or individually chosen background bitmaps [11] designed for easy and definite recognition by users.

In [9], the authors proposed to apply synchronized random dynamic boundaries (SRDs) to distinguish authentic parts of the browser GUI from rendered content received from a server by changing the boundary colors of the real GUI pseudo-random and unpredictable for remote attackers. In [10], the authors introduced the concept of the trusted credential area (TCA) which visualizes the authenticity of a web site by means of extended graphical credentials (e.g. brand logos, icons, seals). In [11], the authors proposed to authenticate browser's secure connection indicators (BSCIs) by applying the concept of personalization with individually chosen background bitmaps. In case of a browser spoofing attack, the information displayed by the browser, and the information displayed in the independent authenticated BSCI will differ, and the visual spoofing attack will be noticed by the user. In [15], the authors proposed the scheme, which is named as "dynamic security skins", with a trusted window in the browser dedicated to username and password entry. If the visual hash which is displayed by the trusted password window does match the website background, it means that a remote attack does not happen. To achieve mutual authentication of the user and the server, the authors implement the Secure Remote Password Protocol (SRP), which is a existing verifier-based protocol, that is required integrated into the SSL/TLS protocol.

However, these countermeasures are not satisfactory due to the following reasons: (1) User's burden: they require mandatory effort of users, such as attention to the change of boundaries, graphic credentials, and the background bitmaps in web browser at any moment; (2) GUI interference: the GUI may be changed in a non-friendly way; (3) machine-dependent: some previous methods require a personal image or image database on the computer. If a user visits an Internet bank at an Internet Cafe, it is impractical for the Internet Cafe to provide personal image or image database.

In this paper, we propose a countermeasure to web-spoofing, which is referred to as automatic detecting security indicator (ADSI). ADSI consists of three modules whose tasks are, respectively, those of: creating a random indicator, embedding the random indicator, and detecting the random indicator. ADSI makes sure that the present web page is trustworthy for SSL-protected server, and gives an alarm when a faked page is identified.

Compared with the previous countermeasures, our solution has the following advantages : (1) Relax user's burden: our solution aims to remove the user's burden by automating the process of detection and recognition of web-spoofing for SSL-based communication. Almost all the previous methods require that the user pay attention to the changes of the browser; (2) Ease of use: this scheme has little intrusion on the browser while other countermeasures may need to disable JavaScript, pop-up windows (e.g., [12]), or change the color of the boundaries (e.g., [9]), or implement a trusted window in the browser dedicated to username and password entry (e.g., [15]; (3) Machine-independent: This scheme can be implemented in any trusted PC at an Internet Cafe without requiring personal folders with individually chosen background bitmaps (e.g., [11], [15]).

In this paper, we analyze the problems in those anti-spoofing countermeasures, and introduce the attack on SSL-protected trusted communication. We also propose a countermeasure to web-spoofing, which is referred to as automatic detecting security indicator (ADSI).

The organization of the paper is as follows. In Section 2, we briefly review related works. Section

3 describes the trust model. Section 4 designs the attack on trusted communication. Section 5 proposes the scheme of ADSI and addresses the performance of ADSI. Section 6 describes the implementation of ADSI. Section 7 describes the analysis, and discussion on the countermeasures of the trust model. At last, we conclude the paper and discuss some future directions in Section 8.

2. Related Work

The SSL-protected transaction process is as follows. The SSL-enabled server owns a certificate issued by a certificate authority, and responds to the client's requests. When a user requests a secure page via a web browser, 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 browser status line if the server is authenticated. Additionally, if the user clicks the security lock, the security information such as the server certificate information will be shown on the data area of the popup window. Nonetheless, the user may be cheated by present web-spoofing attack.

To mount a web-spoofing attack, the attacker would lure the client to accept malicious packets so as to sit in the middle between the target server and client. The connection could be redirected to the attacker's site pretended to be a trusted server without any client notification. As a result, the attacker is able to insert, delete, and tamper the communication data. The attacker may also feed the browser with the faked web content with a manipulated certificate. In this way, a malicious web server is set up to communicate with the client "securely". Although bogus pages are shown to the user, the GUI looks like the same as the genuine one when a web-spoofing attack is mounted successfully.

Felten et al [2] 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 [2] 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 to 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 [16] 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 [8] 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 [10] and Paoli et al [11] adopted the patch method.

Ye et al [9] 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-spoofing attack is under way. For the small screen devices (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.

Herzberg et al [10] proposed the concept of the trusted credential area (TCA). The authors suggest creating a "trusted credentials area" as a fixed part of the browser window. The trusted credentials areas are unspoofable areas in a browser's user interface, which visualize the credentials from the web site, such as logos, icons, and seals of the brand, that have been certified by trusted certificate authorities. A strength of the solution is that it does not rely on complex security indicators. However, careful consideration must be given to the design of an indicator for insecure windows so that spoofed credentials can be easily detected. It is not clear how logos will be certified, and how disputes will be resolved in the case of similar logos. This method is costly for a certification authority to verify that certified logos are sufficiently distinct. Unfortunately, in some cases, users may be confused by the imitational brand logos. To detect this attack, they must be able to inspect the certificate, and to distinguish the domain name of the real web site from the spoofing site.

Adelsbach et al [11] proposed to implement and combine all concepts in an adaptive web browser toolbar, which summarizes all relevant information, and allows the user to get this crucial information at a glance. Since this toolbar is a local component of the user's system, a remote attacker cannot access it by means of active web languages. The advantage of this implementation is that a user has a permanent and reliable overview about the status of his web connection. Once a user has personalized the browser's GUI, the user achieves sufficient security against visual spoofing attacks. The user only has to verify the web browser's personalization and the certificate information, which is always displayed. A disadvantage of the toolbar described above is that its implementation depends on the underlying browser, as it must be integrated in the web browser's user interface. At each login, the user must recognize his personal image.

Dhamija et al [15] proposed the scheme "dynamic security skins", that allows a remote web server to prove its identity in a way that is easy for a human user to verify and hard for an attacker to spoof. The browser extension provides a trusted window in the browser dedicated to username and password entry. The scheme allows the remote server to generate a unique abstract image for each user and each transaction. This image creates a skin that automatically customizes the browser window or the user interface elements in the content of a remote web page. To authenticate content from the server, the user can visually verify that the images match. Given an initial seed, Random Art generates a random mathematical formula that defines a color value for each pixel in an image [17]. The image generation process is deterministic and the image depends only on the initial seed. The advantage is that the user has to recognize only one image and remember one low entropy password, no matter how many servers he wishes to interact with. Another advantage is that, to authenticate content from an authenticated server, the user only needs to perform one visual matching operation to compare two images. The disadvantage of this method is that the GUI may be changed in a non-friendly way, because its implementation depends on the trusted window. This method also requires personal image, which can not be used in the Internet Cafe. The method requires the user to pay attention to the difference of images between the trusted window and the background bitmaps of the browser. The implementation of the method is based on the secure remote password protocol, this could result in a drawback, because it requires the integrating of SRP with SSL/TLS to the web server side.

3. Trust model

It is assumed that ADSI relies on a trusted computer, which is a safe environment without spyware or virus. This assumption is widely accepted by other schemes [10] [11], and also reasonable since it is impossible to provide security function in a compromised computer. A trusted PC is different from a personal PC. Concretely, a user has partial control of a trusted PC, but total control of a personal PC. For example, a home computer is a personal PC, while a non-compromised computer at an Internet Cafe is a trusted PC, since a normal user can not change the configuration of the computer. Thus, the previous schemes (e.g., [11]) are only applicable to personal computers since the computers are customized to link security signs with individually chosen background bitmaps in the personal folders.

With respect to Fig.1, the trust model in ADSI has four parties: user, web client, web server, and attacker. In this model, the user initiates a web client (e.g., Netscape Navigator or Microsoft Internet Explorer) to access a web server, so as to download content or a web page from the server. The client may reside either on a trusted PC at an Internet Cafe or on a personal PC. The attacker can launch any spoofing attacks between the server and the client.

The trust path from the user to the web server have four sub-paths: Sub-path 1, user to PC; Sub-path 2, PC to web browser; Sub-path 3, web browser to web content; and Sub-path 4, web content to web server.

- ◇ [Sub-path 1]-user to PC: The user needs to trust the PC in a transaction process. As mentioned above, the trusted PC was not infected with any virus or spyware, such as a keystroke logger, and provides the platform for our ADSI.
- ◇ [Sub-path 2]-PC to web browser: The browser needs to also be trusted. Most attackers manipulate spoofed browsers to lure the user to input passwords, since this method is easily implemented and effective. Protecting

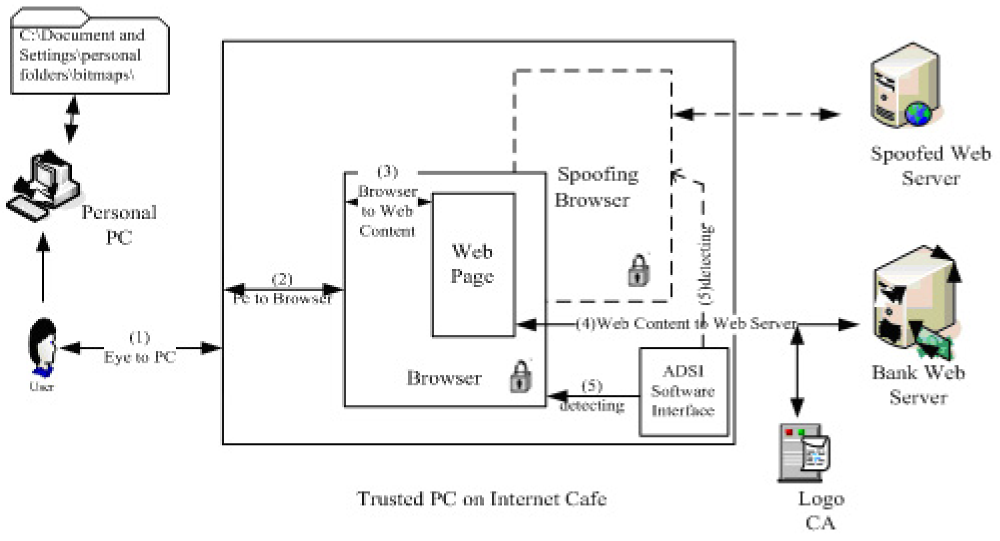


Fig. 1. Trust Model.

the browser from spoofing, especially the SSL-enabled browser, is the major concern of the anti-spoofing community.

◇ [Sub-path 3]-web browser to web content: Even with the genuine browser, the rendered web content may still be counterfeited. Security signs which are unknown to the attacker, e.g., personal icons instead of the padlock (which might be a fake one), are needed to visually remind the user of the authenticity of the web content.
◇ [sub-path 4]-web content to web server: Rarely mentioned, yet not unimportant, all of the security signs may indicate that the web server is authenticated with SSL protected protocol, only the web server is not the one the user desires (e.g., the user may be confused by two similar URLs: `www.citibank.com` and `www.citibanks.com`. While the latter has registered at some CA with a genuine certificate). Thus, it is the responsibility of the user to check the authenticity of the URL.

4. Attacks on Trusted Communication

This section designs the attack on the sub-path2 of the trust model in SSL protected communication.

With reference to Fig.1, the attacker can insert, delete, and tamper the communication data. He may feed the browser with the faked web content and a manipulated certificate. In sum, a spoofed web server is set up to communicate with the client securely. 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 box should be pop up to convince the user that all the security information is correct.

Technically, this bogus interface is generated after the client accepts a malicious HTML file (see Table I), which creates a new window without a status line. The new window is split into two frames. One is named the *upperFrame*. The *upperFrame* showing the web page `attacker.html` of the target site, is provided

Table I. 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>
```

Table II. 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>
```

Table III. 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>
```

Table IV. 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>
```

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 icons 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`.

To cheat the careful users, `lock.html` provides the security lock icon, as well as its actions. Table II 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.

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 an object actually, 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: the "`top`" property specifies its position so as to move the layer; "`visibility`" controls whether the layer is displayed;"`id`" identifies the layer. Table III sets up the layer to forge the certificate window. Table III is the code that generates 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 III is appended to the server web, as well as the jar file "`Spoof.jar`".

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 II) indicates that the function `showLayer()` will make the layer visible when the *onClick* event occurs. The code in module `showLayer()` in Table IV finds the document at first, then finds the layer, based on layer `id` value "*Layer1*" attribute, which is defined in Table III. 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 not destroyed, when closing the certificate window.

Table V. 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);}
});
```

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 V illustrates this code.
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 on Trusted Communication

In this section, we describe the Automatic Detecting Security Indicator (ADSI) for preventing web-spoofing in detail.

As a countermeasure to web-spoofing, ADSI (Automatic Detecting Security Indicator) is a part of the browser (e.g., Microsoft Internet Explorer). When a user starts a browser, ADSI is initiated at the same time. ADSI includes three modules: creating the random indicator, embedding the random indicator, and detecting the random indicator.

Before we describe the modules of ADSI, we define a logic concept as the Secure Browsing Session (SBS). A user initiates SBS on a trusted PC (Fig. 1). Within an SBS, the user shall start a browser, visit some web sites, and enjoy the surfing experiences. ADSI exists for the whole lifetime of the session. In other words, the existence of ADSI implies an active SBS. Furthermore, the user is not limited on the usage of SBS in certain PC. Notice that one SBS maps to one ADSI only, yet we allow the user to open many SBSs on the same PC. To simplify the description, we discuss a single SBS with a single ADSI.

An SBS is typically started with an activation of a browser. ADSI is initiated as a parallel process. Then ADSI creates the random indicator. When the SBS sends the secure web page request, the SBS calls the embedding of the random indicator module at the same time. Whenever there is any new event, such as starting an new HTTPS request, ADSI will be triggered on detecting the changes of status. ADSI snapshots the proper region of the position, and compares it with the indicator. If the two pictures are matching, ADSI will keep silence. Otherwise, ADSI will report a web-spoofing alarm. Finally, the user needs to take care of closing his SBS at the end of the session to erase his sensitive information while accessing some security related web sites. ADSI ends with the closing of the session. The schematic representation of a sequence of process in SBS and ADSI is shown in Fig.2.

5.1. Creating the random indicator

To automatically identify authenticated web pages and detect bogus pages, ADSI can generate random image. For instance, an indicator image is produced with random seed such as URL, screen data, system time and file records, etc[17]. In subsection 6.2, we choose another method to generate a random indicator. The random indicator is integrated into the ADSI.

5.2. Embedding the random indicator

After the random indicator is generated, it will be embedded into a random position in the browser to indicate a secure page. The random embedding makes sure that a faked popup browser window is not able to predict the position and replicate the image.

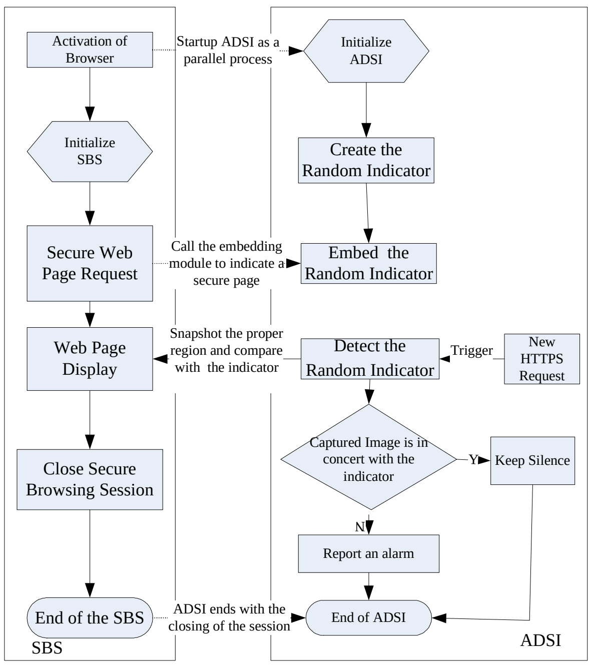

Activation of Browser
Initialize SBS
Secure Web Page Request
Web Page Display
Close Secure Browsing Session
End of the SBS
SBS
Startup ADSI as a parallel process
Call the embedding module to indicate a secure page
Snapshot the proper region and compare with the indicator
ADSI ends with the closing of the session
Initialize ADSI
Create the Random Indicator
Embed the Random Indicator
Detect the Random Indicator
Trigger
New HTTPS Request
Captured Image is in concert with the indicator
Y
Keep Silence
N
Report an alarm
End of ADSI
ADSI

Fig. 2. The schematic representation of a sequence of process in SBS and ADSI.

5.3. Detecting the random indicator

Whenever there is any new action of the web browser, i.e., connecting with a new web site, or starting an SSL connection request, ADSI will be triggered on checking the changes of status. ADSI will also trigger the checking process if an HTTPS is shown in the URL bar. To check the authenticity, ADSI will check the indicator at its proper position on the browser, e.g., the Up-Right region of the browser where the image may occur. Please recall that the adversary cannot predict the region of the position, but ADSI knows the region. To detect the image, ADSI may snapshot the proper region, and compare it with the indicator. If the captured image is not in concert with the indicator, ADSI will report a web-spoofing, e.g., an alarm with a speaker or institutive sequence. Otherwise, ADSI will keep silence.

6. Implementation

We choose Mozilla 3.5, which is an open source browser, to implement visualization of ADSI .

6.1. Mozilla user interface and architecture

The visualization of ADSI is implemented as a Mozilla extension.

Mozilla has configurable and downloadable chrome. In fact, most of Mozilla's windows (and dialogs) will be described using this mechanism. XML user-interface language (XUL) is the name of the language in which these UI (User-interface) descriptions are built [18].

In Mozilla, XUL is handled much in the same way that HTML, or other types of content are handled. When you type the URL of an HTML page into the browser's address field, the browser locates the web site and downloads the content. The Mozilla rendering engine takes the content in the form of HTML source code and transforms it into a document tree. The tree is then converted into a set of objects that can be displayed on the screen. Cascading Style Sheets (CSS), images, and other technologies are used to control the presentation. XUL functions in much the same way.

Mozilla provides a method of installing content locally, and registering the installed files as part of its `chrome` system. This allows a special URL form to be used which is called a chrome: `//URL`. By accessing a file using a chrome URL, the files receive elevated privileges to access local files, access preferences and bookmarks, and perform other privileged operations.

There are usually three different parts to a chrome package, although they are all optional. Each part is stored in a different directory. These three sets are the content, the skin, and the locale. A particular package might provide one or more skins and locales, but a user can replace them with their own. In addition, the package might include several different applications, each accessible via different chrome URLs. The packaging system is flexible enough so that you can include whatever parts you need, and allow other parts, such as the text for different languages, to be downloaded separately.

This chrome package registration is the way Firefox extensions are able to add features to the browser. The extensions are small packages of XUL files, JavaScript, style sheets and images packed together into a single file. This file can be created by using a ZIP utility. When the user downloads it, it will be installed onto the user's machine. It will hook into the browser using a XUL specific feature called an overlay which allows the XUL from the extension, and the XUL in the browser, to combine together. To the user, it may seem like the extension has modified the browser, but in reality, the code is all separate, and the extension may be uninstalled easily.

6.2. Design considerations for implementation of ADSI

In essence, there are four main steps to look into in the implementation of ADSI.

Firstly, the image is displayed at a random place in the chrome area of Mozilla when the web document request is of `https` request. Originally, it is considered `menu-bar`, `navigator toolbar`, and `status-bar` to display the image. But, as the program progresses, it is found that ADSI is inaccurate when displaying image at `status-bar`. It is because the images is deformed when shown at `status-bar`. At last, we decide 9 random places to display image, which are before or after `menubar-items`, before the `back-forward-button`, `reload-button`, `stop-button`, `home-button`, `urlbar-container`, `search-container`, and after `search-container`.

Secondly, we choose which image to be displayed at the random place. At this stage of infancy, we get 1 among 10 images by using random algorithm. In order to show the image, we add a toolbar button to a Toolkit application using overlays, and assign `myImage` as the id to this button. XUL elements in the mozilla chrome are assigned an id for easy accessibility by other elements.

Thirdly, after the browser requested page have been loaded, we grab the image, which is displayed at the certain place of the browser chosen at the first step.

Finally, we compare these two images, one is the source image which is chosen at the second step, and another is grabbed at the third step. If these two images are the same, ADSI will do nothing, otherwise, it will report an alarm.

6.3. GUI implementation

We add the button `myImage` at the chrome area of mozilla 3.5. It can be accomplished by adding `myImageoverlay.xul`. If the web document request is of `https`, the button `myImage` is displayed at a random place of the chrome area, otherwise the button `myImage` is removed from chrome.

We can receive the message when the browser makes any request by using a function `window.addEventListener()`. When the mozilla makes a `https` request, the location of the button `myImage` is replaced, and the image is changed. These behaviors are written into browser-internal JavaScript code, which is read by the XUL

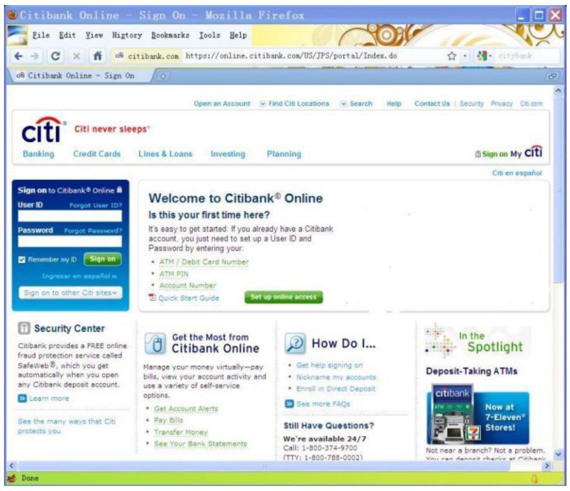

Citibank Online - Sign On - Mozilla Firefox
File Edit View History Bookmarks Tools Help
citibank.com https://online.citibank.com/US/JPS/portal/Index.do
Citibank Online - Sign On
Open an Account | Find Citi Locations | Search | Help | Contact Us | Security | Privacy | Citi.com
citi Citi never sleeps
Banking | Credit Cards | Lines & Loans | Investing | Planning | Sign on My citi
Citi en español
Sign on to Citibank® Online
User ID Forgot User ID?
Password Forgot Password?
Remember my ID | Sign on
Ingresar en español »
Sign on to other Citi sites
Welcome to Citibank® Online
Is this your first time here?
It's easy to get started. If you already have a Citibank account, you just need to set up a User ID and Password by entering your:
- ATM / Debit Card Number
- ATM PIN
- Account Number
Quick Start Guide | Set up online access
Security Center
Citibank provides a FREE online fraud protection service called SafeWeb®, which you get automatically when you open any Citibank deposit account.
Learn more
See the many ways that Citi protects you
Get the Most from Citibank Online
Manage your money virtually—pay bills, view your account activity and use a variety of self-service options.
- Get Account Alerts
- Pay Bills
- Transfer Money
- See Your Bank Statements
How Do I...
- Get help signing on
- Nickname my accounts
- Enroll in Direct Deposit
See more FAQs
Still Have Questions?
We're available 24/7
Call: 1-800-374-9700
(TTY: 1-800-788-0002)
In the Spotlight
Deposit-Taking ATMs
citibank Now at 7-Eleven® Stores!
Not near a branch? Not a problem.
Done

Fig. 3. Logon page of an Internet banking system with embedded `myImage` on the menu-bar generated by ADSI

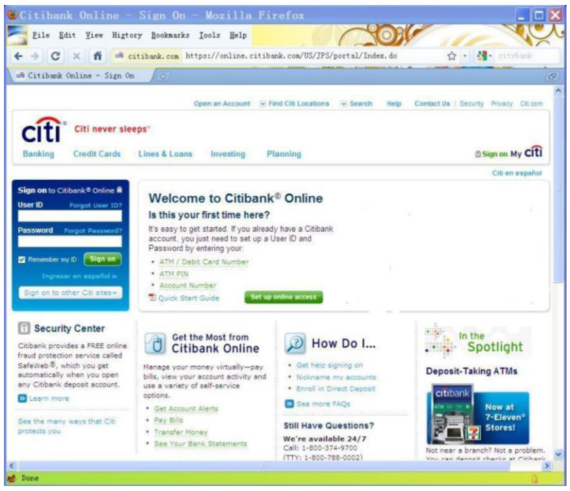

Citibank Online - Sign On - Mozilla Firefox
File Edit View History Bookmarks Tools Help
citibank.com https://online.citibank.com/US/JPS/portal/Index.do
Citibank Online - Sign On
Open an Account | Find Citi Locations | Search | Help | Contact Us | Security | Privacy | Citi.com
citi Citi never sleeps
Banking | Credit Cards | Lines & Loans | Investing | Planning | Sign on My citi
Citi en español
Sign on to Citibank® Online
User ID Forgot User ID?
Password Forgot Password?
Remember my ID | Sign on
Ingresar en español »
Sign on to other Citi sites
Welcome to Citibank® Online
Is this your first time here?
It's easy to get started. If you already have a Citibank account, you just need to set up a User ID and Password by entering your:
- ATM / Debit Card Number
- ATM PIN
- Account Number
Quick Start Guide | Set up online access
Security Center
Citibank provides a FREE online fraud protection service called SafeWeb®, which you get automatically when you open any Citibank deposit account.
Learn more
See the many ways that Citi protects you
Get the Most from Citibank Online
Manage your money virtually—pay bills, view your account activity and use a variety of self-service options.
- Get Account Alerts
- Pay Bills
- Transfer Money
- See Your Bank Statements
How Do I...
- Get help signing on
- Nickname my accounts
- Enroll in Direct Deposit
See more FAQs
Still Have Questions?
We're available 24/7
Call: 1-800-374-9700
(TTY: 1-800-788-0002)
In the Spotlight
Deposit-Taking ATMs
citibank Now at 7-Eleven® Stores!
Not near a branch? Not a problem.
Done

Fig. 4. Logon page of an Internet banking system with embedded `myImage` (beside the home-button) on the navigator toolbar generated by ADSI

file where the button GUI code resides. To change the attribute of `myImage`, we also modify the run time code in `http.js`, which is described in the CSS file `myImage.css`.

We can find that the URL of `list-style-image` is *NULL*, because the image changes as `https` request is sent. After the page that has been requested is loaded, the `http.js` in ADSI will decide where and which image is to be displayed on the chrome randomly. Two screen shots of the end result of two HTTPS requests that have been sent are shown in Fig.3 and Fig.4.

After `myImage` is displayed on chrome, ADSI doesn't stop its work. ADSI will grab the button `myImage` as soon as the image has been displayed. Mozilla divides things into two sets of trees, the content tree, and the layout tree. The content tree stores the nodes as they are found in the source code. The layout tree holds a different tree of nodes for each individual component that can be displayed. The layout tree holds the structure as the nodes are expected to be displayed. The document object model (DOM) is generally used only to get and modify information pertaining to the content or structure of the document. It's relatively simple to determine what kind of content tree node is created for a given element. A XUL element, for example, has a `XULElement` type of content node. XUL provides box objects, which can provide some layout related information. As the name implies, they are available for all box-based elements. The box object provides four properties, x, y, width, and height, for determining the position and size of an element. So we can use `document.getElementById("myImage").boxObject` to get the button `myImage`'s coordinate and size. Since Mozilla allows users to use canvas to draw graphs, and make photo compositions, our software ADSI can draw image of this area name `grabbedImage`. Our software ADSI can get a picture where size and pixel data is the same as the image `myImage`'s. If phishing attacks do not happen, the grabbed picture is the same as the source image which only knows by program code, which is named as `sourceImage`, otherwise, these two images are different.

The `CanvasPixelArray` object can be accessed to look at the raw pixel data. Each pixel is represented by four one-bytevalues (red, green, blue, and alpha, in that order). Each color component is represented by an integer between 0 and 255. Each component is assigned a consecutive index within the array, with the top left pixel's red component being at index 0 within the array. Pixels proceeds from left to right, then downward, throughout the array. The `CanvasPixelArray` contains $height \times width \times 4$ bytes of data, with index values ranging from 0 to $(height \times width \times 4) - 1$. We can get the value of each pixel of `grabedImage` and `sourceImage`, then compare two values. If all the pixels of the two images are the same, ADSI keeps silence and the new SSL session can be regarded as a secure session.

If the spoofing attacks don't happen, the grabbed picture is the same as the source image. As the attacks happen, which is presented in section 4, the attack on trusted communication can fake a new bogus window, and its method. The adversary uses browser features to make it appear as if the browser displays the SSL-protected victim web page, while in fact it is displaying a cloned page. If this spoofing attack happens, ADSI will popup a window with alarm.

The adversary uses browser features to make it appear as if the browser displays the SSL-protected victim web page, while in fact it is displaying a cloned page. If this spoofing attack happens, ADSI will popup a window with sound alarm to attract the user that s/he is suffering from spoofing attacks. In fact, it turns out that many existing web sites such as `Citibank` require sensitive information such as user ID and passwords. For example, the attacker clones `Citibank` login screen, which is in front of a larger one, that is simply the regular `Citibank`. It is assumed that the attacker does not use our implementation of ADSI. ADSI is triggered and the window with sound alarm is popped up. The screen shot of this situation is shown in Fig.5. It is showed that there isn't `myImage` button on chrome (either on menu-bar or navigator toolbar), and the real web page, which has the button `myImage`, is behind of the spoofing page. As the attacks happen, which is presented in section 4, the bogus reference window will overlay the original reference window. Although that myImage.xpi can be downloaded from a web site and the source code can be read by any adversary, the image gallery of myImage.xpi can be integrated into the software or substituted by the user, and the choice, places of displaying image are also computed using the random algorithm. The potential attack can be defended effectively with the use of the customization and random algorithms.

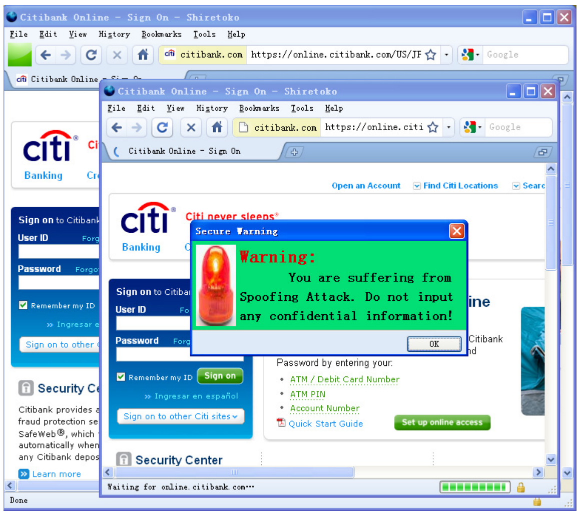
6.4. Installation

Because the mozilla browser allows the user to make a custom toolbar button, the installation of `myImage` requires that the user having at least 9 random places to display image. It can be installed on Mozilla just dragging `myImage.xpi` (available on `http://trust.csu.edu.cn/faculty/~csqifang/software.html`) into browser window.

7. Discussion

7.1. Automatic detection

As its most important property, ADSI automatically detects and recognizes the web-spoofing for SSL-enabled communication . ADSI takes into account the



Secure Warning

Warning:

You are suffering from Spoofing Attack. Do not input any confidential information!

Waiting for online.citibank.com…

Fig. 5. Logon page of an Internet banking system and spoofing page with sound alarm generated by ADSI

limited skills for a user to detect a browser's spoofing attack. Previous research has shown that even under normal conditions, it is difficult for average users to determine whether a browser connection is secure [15] [19]. It is indicated that intentional spoofing attacks make this an even more challenging task for users.

In subsection 6.2, it is shown that the implementation of creating the random indicator, embedding the random indicator and detecting the random indicator are all automatic. The choice of random image and displayed random place are all performed by the source code of `myImage.xpi`. The following process of performing matching operation to compare the grabbed images and the indicator is also performed by the source code of `myImage.xpi` automatically. If the captured image is not in concert with the indicator, the source code of `myImage.xpi` will report an alarm with a speaker or institutive sequence. Our solution implements automatic detection. The user only needs to install the myImage.xpi. ADSI relaxes the burden of the user effectively.

7.2. Ease of use

Some previous schemes (e.g., [12]) disable Java Script, pop-up windows, and configuration of the status bar so as to prevent web-spoofing. These schemes are simple and effective but sacrifice some important properties of web applications. In addition, it is not easy to expect the naïve users to understand the functions and configuration of a web browser's status bar, location bar, and so on. We believe most users can be reminded by the sensitive alarm report or warning sequence. After displaying the alarm information, ADSI allows users to verify whether the used web page is trustworthy of not. In addition, our scheme has little intrusiveness on the user when the page is authentic, while others may change the GUI interference in a non-friendly way (e.g., [12] [15]). The implementation of the dynamic security skin [15] is based on the secure remote password protocol, which is a drawback, because its implementation requires the integrating Secure Remote Protocol with SSL/TLS to the web server. The change of the web server represent an obstacle to widespread deployment. Our scheme need not do any change of the web server.

7.3. Machine-independent

Some previous countermeasures need to configure the computer, or require the personal image or image database on the personal PC, but ADSI can be executed in any trusted computer. Clearly, it is inappropriate to require user only using personal PC, since a personal image or image database is not always available on those PCs such as in Internet Cafe or airports.

7.4. Protected Paths

Based on the trust model in Fig.1, there are several sub-paths in the communication. Different countermeasures may be deployed in different paths.

TCA [10] protecting Sub-path 4 in our trust model, depends on a trusted third party called Logo Certification Authority (LCA), to visualize the authenticity of a web server by means of extended graphical credentials (e.g. brand logos, icons. seals). Apparently, this requirement level is strict, and the overhead of "certification" load is significant compared with the other two solutions. Besides its costs, it has distinct leakages. Some attackers can have gone through the effort of registering a real certificate for their logos of rogue spoofing sites, to imitate the original web sites. In this case, users usually feel confused by alike logos, and can not rely simply on the presence of a TCA certificate.

Personal Authentication solution [11], as in Sub-path 3 of our trust model, requires that PCs have personal folders with individually chosen background bitmaps. This is not inappropriate for users using a PC at an Internet Cafe. However, the requirement of users' creating and remembering all the individual images for web servers imposes mandatory effort of the users. Our solution selects to automate this process of detection and recognition to remove the burden from the users.

Our ADSI solution defends against spoofing attacks on SSL-enabled web servers in Sub-path 2 of our trust model. ADSI can defend against the browser spoofing attack with the lowest secure requirement level, which only requires the PC is trusted. Our solution requires neither Logo Certification Authority, nor the personal folders with individually chosen background bitmaps. Our solution can fully defend the attacks described in Section 4.

7.5. Extension

Since ADSI may randomly choose a picture, and embed it into the current web browser at the random place, an adversary cannot predict it, thus the attacker can not spoof the randomly embedded indicator. Compared with a familiar image of the user [11], a randomly embedded image is not quite attractive to the user. The ignorance of the user can be compensated by the automatic checking mechanism of our ADSI, as it will alarm the user in any way upon detecting a mismatch. To this end, we propose a simple and effective way in protecting naïve users from pop up browser spoofing attacks. As the spoofing techniques evolve, and are used jointly, we can see that our method can not defend against them without accommodating other security mechanisms such as personal image [11] and TCA [10]. Fortunately, aforementioned ADSI does not conflict with any other security means, but compensates with them. Recall the trust model of Section 3.We secure the trust path from PC to Browser, while [11] secures it from browser to web content, and [10] from web content to web server. Together, they may form an integrated solution providing strong security.

8. Conclusion

In this paper, we analyzed the problems in web-spoofing. Spoofing vulnerabilities were known years ago, yet no suitable solutions proposed were efficient enough for protecting the non-cautious users. As analyzed in our trust model, different security mechanisms are designed to defend against special kinds of spoofing attacks. The attackers tend to use their attacking skills for launching complex attacks with advanced strategy. To battle against them, security designers must work together on providing syndicated systematical defending technologies to benefit the users.

We proposed the countermeasure called ADSI, an automatic anti-spoofing tool that can not only function independently, but also combine other anti-spoofing techniques to form more powerful defences. ADSI relaxes user burden by automating the process of detection and recognition of the web-spoofing for SSL-enabled communication. Our solution decreased intrusiveness on the browser, while other countermeasures may disable Java Script, pop-up windows or change the color of the boundaries. Our solution can defend the browser spoofing attacks with the lowest secure requirement level, which only requires the PC is trusted, which is described in our trust model. Our solution requires neither Logo Certification Authority, nor the personal folders with individually chosen background bitmaps. In our future work, we must consider how to make the image gallery of `myImage.xpi` more flexible. All the source codes should be in stand alone rather than in the Mozilla's extension.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant Nos. 90718034 and 60773013, Hunan Provincial Natural Science Foundation of China No. 09JJ4031, the Program for New Century Excellent Talents in University (NCET-06-0686), and the Program for Changjiang Scholars and Innovative Research Team in University under Grant No. IRT0661.

References

1. Freier A.O., Kariton P., Kocher P.C. 1996. *The SSL Protocol: Version 3.0. Internet draft*. Netscape Communications.
2. Felten E.W., Balfanz D., Dean D., Wallach D.S. 1997. *Web Spoofing: An Internet Con Game.* Proceedings of the 20th National Information Systems Security Conference, Baltimore, USA
3. Tamara D. 2006. *Why spoofing is serious: Internet fraud.* Communications of the ACM 49:1010, pp.77-82, Association for Computing Machinery.
4. Wu M., Robert C. M., Little G. 2006. *Web wallet: preventing phishing attacks by revealing user intentions.* Proceedings of the second symposium on Usable privacy and security. pp.102-113
5. Emigh A. 2005. *Online Identity Theft: Phishing Technology, Chokepoints and Countermeasure*. ITTC Report on Online Identity Theft Technology and Countermeasures. October 3.
6. Chou N., Ledesma R., Teraguchi Y., Boneh D. and Mitchell J. 2005. *Client-side defense against web-based identity theft.* In Proc. 11th Annual Network and Distributed System Security Symp. San Diego, CA, February 5-6, pp.119-128.

7. Jagatic T.N., Johnson N.A., Jakobsson M., and Menczer F. *Social phishing.* Communications of the ACM, 2007
8. Florencio D., Herley C. 2007. *Evaluating a trial deployment of password re-use for phishing prevention.* Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit. Pittsburgh, Pennsylvania,ACM.
9. Ye Z.S., Smith S., Anthony D. 2005. *Trusted Paths for Browsers.* ACM Transactions on Information and System Security (TISSEC). Volume 8, Issue 2, pp.153-186.
10. Herzberg A. and Gbara A. 2004. *TrustBar: Protecting (evenNaive) Web Users from Spoofing and Phishing Attacks.* Cryptology ePrint Archive: Report 2004/155.
11. Adelsbach A., Gajek S., and Schwenk J. 2005. *Visual Spoofing of SSL Protected Web Sites and Effective Countermeasures.* Proceedings of Information Security Practice and Experience '2005, LNCS 3469, pp.204-216.
12. Li T.Y., Wu Y.D. 2003. *Trust on Web Browser: Attack vs. Defense.* Proceedings of the International Conference on Applied Cryptography and Network Security, LNCS 2846, pp.241-253.
13. Tygar J.D., Whitten A. 1996. *WWW Electronic Commerce and Java Trojan Horses.* Proceedings of the Second USENIX Workshop on Electronic Commerce.
14. Wang Y., Agrawal R., Choi B.Y. 2008. *Light Weight Anti-Phishing with User Whitelisting in a Web Browser.* 2008 IEEE Region 5 Conference
15. Dhamija R., Tygar J.D. 2005. *The Battle Against Phishing Dynamic Security Skins.* Proceedings of the ACM Symposium on Usable Security and Privacy, July 2005.
16. Lefranc S. and Naccache D. 2003. *Cut and Paste Attacks with Java.* Proceedings of the 5th International Conference on Information Security and Cryptology, LNCS 2587, pp.1-15.
17. Bauer, Anrej, Random Art, http://gs2.sp.cs.cmu.edu/art/random/
18. Mozilla Chrome http://www.mozilla.org/xpfe/ConfigChromeSpec.html
19. Friedman B., Hurley D., Howe D., Felten E., Nissenbaum H. 2002. *Users' Conceptions of Web Security: A Comparative Study.* CHI 2002 Extended Abstracts of the Conference on Human Factors in Computing Systems, pp.746-747.

Authors' Biographies

Fang Qi is a lecturer at Central South University. She received a B.Sc. degree from the National University of Defense Technology. She received her M.Sc. and Ph.D degrees from Central South University. Her current research areas are network security, mobile computing, and wireless communications.

Zhe Tang is an associate professor at Central South University. He received a B.Sc. degree from the National University of Defense Technology. He received his M.Sc. and Ph.D degrees from Tsinghua University. His research interests include humanoid robotics, intelligent control and computer networking.

Guojun Wang received B.Sc. in Geophysics, M.Sc. in Computer Science, and Ph.D. in Computer Science, from Central South University, China. He is currently an Adjunct Professor at Temple University, USA. Since 2005, he has been a Professor at Central South University, China. He is the Director of Trusted Computing Institute of the University. He has been a Visiting Scholar at Florida Atlantic University, USA, a Visiting Researcher at the University of Aizu, Japan, and a Research Fellow at the Hong Kong Polytechnic University, HK. His research interests include trusted computing, mobile computing, pervasive computing, and software engineering. He has published more than 140 technical papers and books/chapters in the above areas. He is an associate editor, or on the editorial board of some international journals including Security and Communication Networks, Journal of Computer Systems, Networking, and Communications, and International Journal of Multimedia and Ubiquitous Engineering. He has also served as guest editor-in-chief or guest editor for some international journals including IEICE Transactions on Information and Systems, The Journal of Supercomputing (Springer), Security and Communication Networks (Wiley), and Journal of Computers (Academy Publisher). He has also served as a general chair, program chair, publication chair, publicity chair, session chair, and program committee member for many international conferences such as ICCCN, GLOBECOM, ICC, WCNC, AINA, HPCC, ATC, ISPA, ICYCS, TrustCom, TSP, and UbiSafe. He is a senior member of the China Computer Federation (2005-), the academic committee member of the YOCSEF of the China Computer Federation (2008-2010), and the chair of the YOCSEF Changsha of the China Computer Federation (2007-2008).

Jie Wu is chair and professor at the Department of Computer and Information Sciences at Temple University. He is an IEEE Fellow. He is on the editorial board of IEEE Transactions on Mobile Computing. He was a distinguished professor in the Department of Computer Science and Engineering, Florida Atlantic University. He served as a program director at US NSF from 2006 to 2008. He has been on the editorial board of IEEE Transactions on Parallel and Distributed Systems. He has served as a distinguished visitor of the IEEE Computer Society and is the chairman of the IEEE Technical Committee on Distributed Processing (TCDP). His research interests include wireless networks, and mobile computing and wireless networks, parallel, and distributed systems, and fault-tolerant systems.