

# Browser User Interface Design Flaws

## Exploiting User Ignorance

Aditya K. Sood, Michigan State University  
Richard J. Enbody, Michigan State University

**Abstract.** A browser is considered to be a functional window to the Internet. It is interface software that serves as a communication medium between the users and the Internet. Sophisticated attack patterns and design flaws in browsers pose serious threats to user security, privacy, and integrity. Recent advancements have shown that browser User Interface (UI) design flaws catalyze the vulnerability exploitation. This paper sheds light on the design flaws in Graphical User Interface (GUI) components of browsers that are exploited by the attackers to trick users to perform rogue operations. In most of the cases, the user is unaware of the attack that results in stealth operations. Thus, user ignorance plays a critical role in successful exploitation of the design flaws.

### Introduction

GUI plays an instrumental role in the success of any browser. GUI enables user control and improves interaction with the browser. GUI is considered a part of the user trust model for all types of software including browsers. Apart from the main browser window, GUI in browsers includes notification bars, status bars, address bars, download dialog boxes, HTTP authentication dialog, and browser objects such as frames, buttons, etc.

Users interact with the GUI components in their routine life jobs. GUI flaws are considered design bugs in which an attacker can circumvent the normal functioning of the browser by running malicious JavaScript. Primarily, GUI bugs in browsers are mostly exploited by spoofing [1] and clickjacking attacks [2]. Spoofing attacks are those kinds of attacks that tamper the UI component of software in order to fool users into performing false operations by exploiting their ignorance. They fail to differentiate between the real and manipulated objects in software. Clickjacking attacks fall into the category of UI redressing attacks in which an attacker embeds a hidden UI object such as buttons, frames, etc. to execute stealth functions that are binded to a real object. For example, an attacker can easily place a hidden button over the real button in a browser window that executes a malicious function when a user clicks it.

Basically, spoofing and clickjacking attacks aim at tampering with and manipulating the functional operations of various browser GUI controls. Apart from this, such attacks exploit the user ignorance to a great extent because users are not able to differentiate between the real GUI object and vice versa. Successful GUI attacks depend a lot on the user awareness about the browser controls and their integrity. It is a major concern because exploitation of GUI design flaws can severely impact the user trust thereby resulting in the loss of integrity. This paper discusses design flaws in browsers related to GUI components and how they are exploited by tricking the user.

### HTTP Authentication Dialog Spoofing

Many browsers require HTTP-based authentication in which users have to provide a set of credentials to access the resources. In general, if a resource is protected, the server sends a particular HTTP response to the browser based on which the browser initiates a dialog authentication process. It is one of the main characteristics of browsers to handle HTTP authentication. Every single HTTP authentication process has a realm value associated with it. In general, the realm value is a string that shows the domain name on which resource is protected. The realm value also provides a user supplied string for identity purposes. A user can check the domain name and provide his credentials to gain access to the server. However, recent vulnerabilities have shown the fact that it is possible to manipulate the authentication dialog box. Users are unable to differentiate among the origins of authentication dialog. A dialog box may look real and authentic but it can be spoofed. This type of flaw in browsers results in the stealing of user credentials without users being aware of the reality. For example: Internet Explorer and Google Chrome inherit this design flaw. A serious design flaw in Google Chrome [3],[4] is that an authentication dialog can be completely spoofed and users are not able to distinguish the difference. A spoofed authentication dialog box is presented as in Figure 1.

Figure 1: Spoofed authentication dialog box in Google Chrome



The spoofed authentication dialog box bedazzles the user. However, it has been noticed that a number of users fall into this trap and provide their authentication credentials as per the realm value shown in the dialog box. This design flaw persists because browsers are not able to handle the realm value passed as a parameter to the authenticated HTTP response header and render it directly in the dialog box. Most browsers do not handle the realm value in an appropriate manner, allowing spoofing attacks.

## URL Obfuscation Flaws

URL obfuscation is one of the most notorious problems noticed in browsers. Continuous efforts have resulted in correction of this design problem in a number of browsers such as Mozilla, Internet Explorer, etc. However, browsers such as Google Chrome still inherit this design bug. In 2008, a design flaw [5] was released in Google Chrome that still persists in recent versions [6]. URL obfuscation is a trick that plays around the designing of URLs with certain meta characters in order to confuse browsers as well as users so that they can be redirected to malicious domain. This is a browser design flaw because browsers are not able to render the URLs appropriately thereby resulting in unauthorized redirection. As a result, the browser can be redirected to a malicious domain that is ready to serve malware.

There can be many combinations based on this pattern. It depends on the inherent design of the browser in interpreting a URL. In general, good practice requires that browsers should raise a warning about the obfuscation in a URL and should be smart enough to present a user with an appropriate choice. Primarily, the user thinks that a destination website is Google.com, but in reality, the user is redirected towards yahoo.com. An obfuscated URL is shown in Figure 2.

In Figure 2, Google Chrome is redirected towards yahoo.com instead of raising a warning or going to google.com. A similar test on Mozilla raises a warning about the URL obfuscation as presented in Figure 3:

After a lot of discussion, Mozilla introduced a security check to show concern with URL obfuscation flaws in browsers.

## Manipulating Browser Status Bars

Browser status bars are used to present the active state of links when a user clicks a hyperlink on a webpage. In general, status bars represent the status of hyperlinks. The mindset behind the design of the status bar is that a user can see the authenticity of domain names and hyperlink. Basically, a user believes that the status bar displays the domain name in the form of a URL and the browser redirects to that page upon clicking. Attackers have exploited this design flaw by spoofing the status bar with JavaScript calls such as window.location or window.href to fool users. However, the URL obfuscation trick can also be used to spoof the status bar. An issue was raised in Internet Explorer [7] about the problem in the status bar. When considering spoofed HTML code, Internet Explorer 7 does not appropriately render the information in the status bar whereas Internet Explorer 8 does not even show any information in the status bar when a mouse is pointed over a hyperlink. This is a serious issue because it is the only way a normal user can scrutinize the authenticity of a hyperlink. Figure 4 shows code that is used to spoof the status bar in Internet Explorer.

Figure 2: URL Obfuscation in Google Chrome

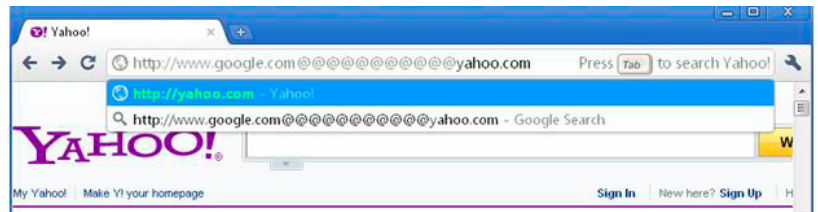
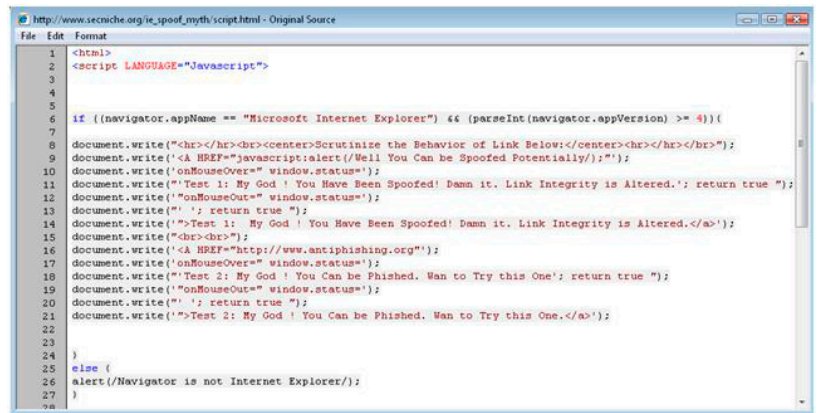


Figure 3: URL Obfuscation Warning in Mozilla Firefox



Figure 4: Custom HTML Code to Spoof Internet Explorer's Status Bar



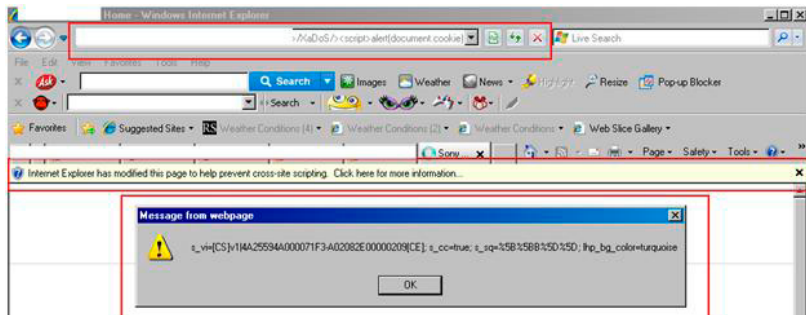
An active Internet Explorer test has been structured here [8]. This is a simple but generic problem in most of the browsers and it is used quite often by attackers to trick users and create a false sense of security.

## Cross Site Scripting Attack Notification Bars – Bypassing Filters

With the advent of new browser security protection mechanisms, reflective Cross Site Scripting (XSS) filters have become a part of the browser architecture. It is an inbuilt protection mechanism that raises XSS attack notification bar for reflective XSS attacks and neutralizes them completely. This is the actual motive behind the designing of XSS filters. However, completely relying on filters as a fool proof protection against XSS attacks creates a false sense of security. The XSS filters in browsers are not well developed and can be bypassed easily to execute successful XSS attacks. Primarily, a user believes that now the browser is secure because of the presence of XSS filters but attackers can exploit the design problem in XSS filters to exploit the trust of users. For example, Internet Explorer released a built-in XSS filter with Internet Explorer 8, but it can be bypassed easily and no notification alert is raised. Moreover, certain stealth XSS attacks were successfully executed in In-

Internet Explorer. However, Internet Explorer's XSS filter raised a notification warning but was not able to sanitize the XSS attacks appropriately. This type of behavior shows the inherent weakness in client-side XSS filters. Moreover, NoScript is considered a very good extension of Mozilla that prevents reflective XSS attacks. However, there are certain bypasses that have been released in it. The good point about this filter is that one can find a lot of updates of this extension. Figure 5 shows a potential attack against the XSS filter in Internet Explorer.

Figure 5: Successful bypass even after XSS notification

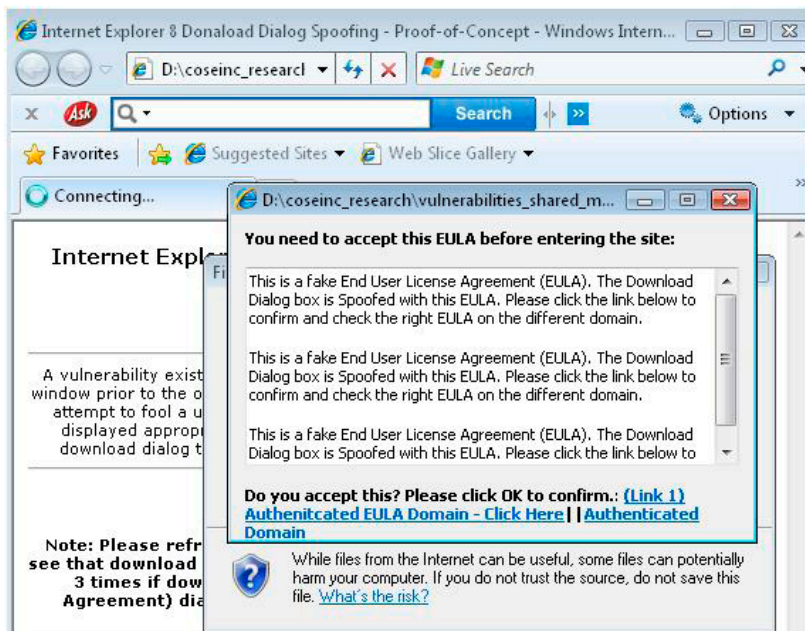


This attack simply projects how the design issues in XSS filters result in exploitation of vulnerability.

### Download Dialog Box Spoofing

Browsers use a download dialog box in order to download a file from a server. This process acts as a notification to the user about the characteristics of the file. The download dialog box is displayed when a user clicks a hyperlink to download a specific file. It is a type of GUI displayed to the user for raising an alert. Attackers are spoofing download dialog boxes to trick users into downloading malicious files instead of authorized files. This attack is triggered on a wide scale to infect user machines with malware. Recently performed tests on Internet Explorer have

Figure 6: Spoofed Download Dialog Box



shown that it is possible to overlap the download dialog box with an unauthorized pop-up window which restricts the functionality of the download dialog box.

Primarily, the overlapped pop-up window forces the user to click some malicious links embedded in it. The pop-up window actually locks the authorized download dialog box and the user fails to download the file directly. This attack is implemented in order to force a user to interact with the rogue pop-up window. In other words, it is a design bug in Internet Explorer that fails to differentiate between the download dialog box and a rogue pop-up window. Figure 6 shows the spoofed download dialog box in Internet Explorer 8.

In the Figure 6 screenshot, a fake End User License Agreement (EULA) pop up window overlaps the authorized download dialog box. This fake EULA window is embedded with malicious links and it locks the download dialog box completely. This attack forces the user to interact with a EULA window prior to downloading the file. In general, users are not aware of these design problems and spoofing tricks which help an attacker to launch attacks successfully. The figure clearly shows one of the serious design bugs in graphical user components in browsers.

### Clickjacking Browser Interface

Clickjacking [9],[10] is a UI redressing attack where an attacker executes malicious functions by playing around with browser UI components. The aim of this attack is to steal sensitive data and extract information about a user's activities in a stealthy manner. Primarily, this attack uses two major UI components in a browser—frames and buttons. The term clickjacking itself points to hijacking mouse clicks in a browser window. In general terms, an attacker designs a transparent UI component such as a button and makes it hidden. When a legitimate user performs a mouse click in a browser window, the hidden button is clicked and it executes the backend command designed by the attacker to perform rogue functions. This attack is considered one of the most sophisticated attacks.

### Solutions

In order to prevent these attacks, here are some measures that can result in mitigating the adverse attacks to some extent, but it is hard to guarantee foolproof solutions:

1. An appropriate browser-based filter should be used while surfing the Internet. For example: NoScript [11] is a good choice. It only works on Mozilla Firefox but it has some built-in capabilities to take control of certain UI redressing attacks such as clickjacking.
2. Browsers should be upgraded regularly and security recommendations must be applied in a timely manner. Most browser software vendors such as Microsoft, Apple, and Mozilla release security advisories about potential vulnerabilities. These security advisories contain an updated fix and patch that should be installed in order to upgrade the requisite browsers. However, if automated updates are enabled, the system is updated regularly against potential threats. A user can also download individual security updates manually from vendor websites.
3. Browser design requires a significant amount of change in the way UI components are handled. However, it becomes hard for the vendors to change UI on a regular basis. This is a para-

## ABOUT THE AUTHORS

dox in the field of browsers, but vendors should take appropriate steps to secure the design interface.

4. Users should not visit those pages that they are not sure of. Sometimes, being paranoid is a good way to be secure. Always think twice about what you click. There are certain client-side browser filters available that help users substantially to make smart decisions if a potential threat is detected. For example, the NoScript plug-in works as an inline component with Mozilla Firefox to strengthen security. It enables the user to surf in a secure manner and raises notification against insecure objects and attacks such as XSS. Other browsers such as Internet Explorer come with built-in client-side protection against XSS attacks. Thus, potential combinations of client-side filters and user awareness can lower the exploitation ratio of vulnerabilities.

5. Users should be aware of the basic attacks on the Internet that can help them in understanding exploitation attempts. There are a number of websites such as Threatpost [12], SecurityFocus [13], and Register, [14] etc. that provide substantial information about new research and attacks.

6. Websites should use frame-bursting scripts [15] to avoid framing of websites. This process is followed in order to avoid loading a website into a frame which is used by a third party. Frame-bursting scripts remove the frame when an attacker tries to load the target website into a frame. This technique avoids the hidden frames used in conjunction to launch clickjacking attacks.

7. A good use of declarative security in HTTP response headers [16,17,18] can circumvent some attacks. This is a potential step in defeating clickjacking attacks. Restricting frames [19] and running them in sandbox is also a good practice.

## Conclusion

We discussed a number of cases of UI design flaws and how they are exploited. During the course of this paper, we have realized that UI is a very critical component of browsers. UI is important because it provides direct functionality to users and helps them to make decisions quickly. However, if UI design flaws are exploited, it becomes much easier to launch attacks as discussed previously. Of course, user ignorance and inappropriate knowledge enhances the chance of exploitation. These design flaws are inherited in browsers to a great extent and it is hard to remove them completely. It is hard to ensure a foolproof solution, but if a reliable set of protective measures is applied, impact can be moderated to some extent. ♦



**Aditya K. Sood** is a security researcher, consultant, and Ph.D. candidate at Michigan State University. He has worked in the security domain for Armorize, COSEINC, and KPMG and founded SecNiche Security. He has been an active speaker at conferences like RSA, Toorcon TRISC, Hacker Halted, EuSecWest, ExCaliburCon, EuSecwest, XCON, OWASP AppSec, Security-Byte, CERT-IN and has written content for HITB Ezine, ISACA, ISSA, Hakin9, and Usenix Login.

**E-mail:** [adi\\_ks@secniche.org](mailto:adi_ks@secniche.org)

**Phone:** 517-755-9911



**Dr. Richard Enbody** is an Associate Professor in the Department of Computer Science and Engineering, Michigan State University. He joined the faculty in 1987 after earning his Ph.D. in Computer Science from the University of Minnesota. His research interests are in computer security, computer architecture, web-based distance education, and parallel processing. He has two patents pending on hardware buffer-overflow protection, which will prevent most computer worms and viruses. He recently co-authored a CS1 Python book, The Practice of Computing using Python.

## REFERENCES

1. Wu, Yongdong, Ma, Di and Sheng, Chnag Xu. "Browser Spoofing". <<http://dspace.lib.fcu.edu.tw/bitstream/2377/1421/1/ce07ics002002000204.PDF>>. 2002
2. Law, Eric. Combating Clickjacking with X-Frame options". <<http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx>>. 23 March 2010
3. Sood, Aditya K. "User Interface Security- Google Chrome: HTTP AUTH Dialog Spoofing through Realm Manipulation ". <<http://zeroknock.blogspot.com/2010/08/google-chrome-http-auth-dialog-through.html>>. 23 August 2010
4. Sood, Aditya K. "Google Chrome: HTTP AUTH Dialog Spoofing through Realm Manipulation (Restated) ". <<http://www.securityfocus.com/archive/1/513243/100/800/threaded>>. 23 August 2010
5. Sood, Aditya K. "Google Chrome MetaCharacter URI Obfuscation Vulnerability ". <<http://www.secureteam.com/windowsntfocus/6L0001FN5S.html>>. 25 November 2008
6. Sood, Aditya K. "Google Chrome URI Obfuscation Vulnerability ". <<http://www.secniche.org/gcuri/>>. 2009
7. Sood, Aditya K. "Internet Explorer 8 - Anti Spoofing is a Myth ". <[http://www.secniche.org/ie\\_spoof\\_myth/](http://www.secniche.org/ie_spoof_myth/)>. 2009
8. Sood, Aditya K. "Internet Explorer 8 - Anti Spoofing is a Myth - Demonstration". <[http://www.secniche.org/ie\\_spoof\\_myth/script.html](http://www.secniche.org/ie_spoof_myth/script.html)>. 2009
9. Hansen, Robert. and Grossman, Jeremiah. "ClickJacking". <<http://www.sectheory.com/clickjacking.htm>>. 12 September 2008
10. Guya. "Malicious Camera Spying using ClickJacking". <<http://blog.guya.net/2008/10/07/malicious-camera-spying-using-clickjacking/>>. 20 October 2008
11. Maone, Giorge. "FAQ's - NoScript Client Side Protection Filter". <<http://noscript.net/faq>>
12. Threatpost, "Latest Computer Security News Portal". <<http://www.threatpost.com>>
13. SecurityFocus, "Security News Portal Website". <<http://www.securityfocus.com>>
14. The Register, "Security News Portal Website". <<http://www.register.com>>
15. Wikipedia. "Frame Killer". <<http://en.wikipedia.org/wiki/Framekiller>>
16. Lawrence, Eric. "IE8 Security Part VII: ClickJacking Defenses". <<http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx>>. 27 January, 2009
17. Sood, Aditya K. and Enbody, Richard J. "Conundrum of Declarative Security in HTTP Response Headers - Lessons Learned". <[http://www.usenix.org/event/collsec10/tech/full\\_papers/Sood.pdf](http://www.usenix.org/event/collsec10/tech/full_papers/Sood.pdf)>. 10 August 2010
18. Coates, Michael. "X-Frame Options". <<http://blog.mozilla.com/security/2010/09/08/x-frame-options/>>. 9 August 2010
19. Lawrence, Eric. "Using Frames More Securely". <<http://blogs.msdn.com/b/ie/archive/2008/01/18/using-frames-more-securely.aspx>>. 18 January, 2008