Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 3, March 2016, pg.27 – 31

Detection of Clickjack Attacks by Employing Agents

Dr. S. Chidambaranathan
Head, Department of MCA, India
s.chidambaranathan14@gmail.com

Abstract— ***Today's world revolves around internet technologies. Numerous internet applications and services are available to the users to make life simpler. In this work, we propose to detect and prevent clickjacking attacks by means of establishing a trustworthy map between the user interface and the web application. This is accomplished by the exploitation of agents. There are three agents proposed in this paper. They are tracking agent, detection agent and action agent. The tracking agent checks for the count and the positional coordinates of the clickable controls and saves it in local memory. The detection agent compares the loaded webpage and the local memory with respect to the count of clickable controls and the positional coordinates. In case of mismatch, the action agent throws warning to the user. The main objective of this work is to provide a secure web application by means of a secure browser. The results of the proposed work are found to be satisfactory.***

Keywords— ***clickjack, agents***

I. Introduction

Today's world revolves around internet technologies. Numerous internet applications and services are available to the users to make life simpler. Cloud computing and internet technologies are closely associated to each other. E-mail, social networks, online banking and online shopping have gained substantial spotlight. On the other hand, due to the advent of cloud computing, all the data are outsourced to cloud service providers. The reason for data outsourcing is to save memory and to escape from the issue of data management. However, the issue of data privacy and security come into play. Thus, it becomes necessary to formulate countermeasures for data protection.

The most serious threat for web applications is cross-site scripting (XSS), which injects malicious scripts into a trustworthy website [1, 2]. These attacks aim at collecting data so as to crack the account of legitimate users, steal cookies etc. The contemporary web applications suffer from clickjacking, which is more serious.

Clickjacking is a security threat for web applications discovered by Jeremiah Grossman and Robert Hansen in 2008 [3]. This technique strives to draw the attention of the user to click on an element of the web page [4]. The attacker employs an iframe, which is a html tag to embed a html document over another document.

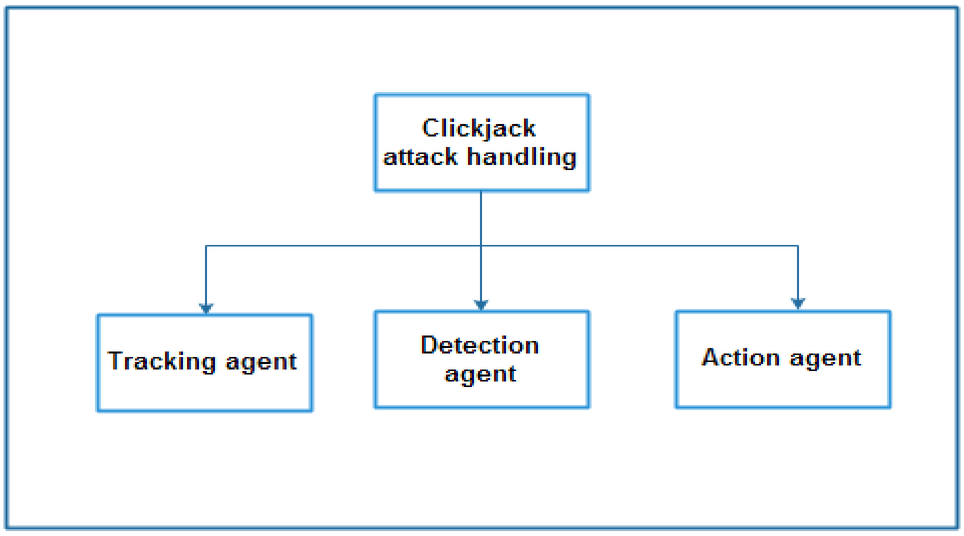

Fig 1: Clickjack attack handling system

To exemplify the concept of clickjacking, consider a normal webpage that possesses a clickable element. On the other hand, the adversary designs a webpage with a clickable element for triggering malicious activity. The malicious activity is done by superimposing the adversary created webpage over the normal webpage. Thus, when a user attempts to click on the normal user interface element, the click operation has been done on the attacker's user interface element.

In this work, we propose to detect and prevent clickjacking attacks by means of establishing a trustworthy route between the user interface and the web application. This is accomplished by the exploitation of agents. The main objective of this work is to provide a secure web application by means of a secure browser. The results of the proposed work are found to be satisfactory.

II. Review of Literature

In this section, we discuss known attacks and defenses for clickjacking, and compare them to our contributions. Below, we assume a victim user is visiting a clickjacking attacker's page, which embeds and manipulates the target element residing on a different domain, such as Facebook's Like button or PayPal's checkout dialog.

A. *Existing clickjack attacks*

We classify existing attacks according to three ways of forcing the user into issuing input commands out of context: (1) compromising target display integrity, the guarantee that users can fully see and recognize the target element before an input action; (2) compromising pointer integrity, the guarantee that users can rely on cursor feedback to select locations for their input events; and (3) compromising temporal integrity, the guarantee that users have enough time to comprehend where they are clicking.

- *Compromising target display integrity*

Hiding the target element: Modern browsers support HTML/CSS styling features that allow attackers to visually hide the target element but still route mouse events to it. For example, an attacker can make the target element transparent by wrapping it in a div container with a CSS opacity value of zero; to entice a victim to click on it, the attacker can draw a decoy under the target element by using a lower CSS z-index [5]. Alternatively, the attacker may completely cover the target element with an opaque decoy, but make the decoy unclickable by setting the CSS property pointer-events:none [6]. A victim's click would then fall through the decoy and land on the (invisible) target element.

Partial overlays: Sometimes, it is possible to visually confuse a victim by obscuring only a part of the target element [7, 8]. For example, attackers could overlay their own information on top of a PayPal checkout iframe to cover the recipient and amount fields while leaving the "Pay" button intact; the victim will thus have incorrect context when clicking on "Pay". This overlaying can be done using CSS z-index or using Flash Player objects that are made topmost with Window Mode property [9] set to wmode=direct. Furthermore, a target element could be partially overlaid by an attacker's popup window [10].

Cropping: Alternatively, the attacker may crop the target element to only show a piece of the target element, such as the "Pay" button, by wrapping the target element in a new iframe that uses carefully chosen negative CSS position offsets and the Pay button's width and height [11]. An extreme variant of cropping is to create multiple 1x1 pixel containers of the target element and using single pixels to draw arbitrary clickable art.

- *Compromising target display integrity*

Proper visual context requires not only the target element, but also all pointer feedback to be fully visible and authentic. Unfortunately, an attacker may violate pointer integrity by displaying a fake cursor icon away from the pointer, known as cursorjacking. This leads victims to misinterpret a click's target, since they will have the wrong perception about the current cursor location. Using the CSS cursor property, an attacker can easily hide the default cursor and programmatically draw a fake cursor elsewhere [12], or alternatively set a custom mouse cursor icon to a deceptive image that has a cursor icon shifted several pixels off the original position [13].

Another variant of cursor manipulation involves the blinking cursor which indicates keyboard focus (e.g., when typing text into an input field). Vulnerabilities in major browsers have allowed attackers to manipulate keyboard focus using strokejacking attacks [14, 15]. For example, an attacker can embed the target element in a hidden frame, while asking users to type some text into a fake attacker-controlled input field. As the victim is typing, the attacker can momentarily switch keyboard focus to the target element. The blinking cursor confuses victims into thinking that they are typing text into the attacker's input field, whereas they are actually interacting with the target element.

- *Compromising temporal integrity*

Attacks in the previous two sections manipulated visual context to trick the user into sending input to the wrong UI element. An orthogonal way of achieving the same goal is to manipulate UI elements after the user has decided to click, but before the actual click occurs. Humans typically require a few hundred milliseconds to react to visual changes [16, 17], and attackers can take advantage of our slow reaction to launch timing attacks. For example, an attacker could move the target element (via CSS position properties) on top of a decoy button shortly after the victim hovers the cursor over the decoy, in anticipation of the click. To predict clicks more effectively, the attacker could ask the victim to repetitively click objects in a malicious game [18, 19, 17, 20] or to double-click on a decoy button, moving the target element over the decoy immediately after the first click [21, 22].

- *Consequences*

To date, there have been two kinds of widespread clickjacking attacks in the wild: Tweetbomb [23] and Likejacking [24]. In both attacks, an attacker tricks victims to click on Twitter Tweet or Facebook Like buttons using hiding techniques described in Section 3.1.1, causing a link to the attacker's site to be reposted to the victim's friends and thus propagating the link virally. These attacks increase traffic to the attacker's site and harvest a large number of unwitting friends or followers.

Many proof-of-concept clickjacking techniques have also been published. Although the impact of these attacks in the wild is unclear, they do demonstrate more serious damages and motivate effective defenses. In one case [25], attackers steal user's private data by hijacking a button on the approval pages of the OAuth [26] protocol, which lets users share private resources such as photos or contacts across web sites without handing out credentials. Several attacks target the Flash Player webcam settings dialogs (shown in Figure 1), allowing rogue sites to access the victim's webcam and spy on the user [18, 19, 27]. Other POCs have forged votes in online polls, committed click fraud [28], uploaded private files via the HTML5 File API [3], stolen victims' location information [17], and injected content across domains (in an XSS spirit) by tricking the victim to perform a drag and drop action [29, 30].

III. PROPOSED APPROACH

This work aims at providing a trustworthy web service by means of a secured web browser. The architecture of the proposed approach relies on three different agent tracking agent, detection agent and action agent. This work is based on the assumption that the attacks happen only after the page load. Overview of the proposed work is presented below.

A. *Tracking agent*

The tracking agent tracks the web pages and the user interface controls. Clickable controls are given much importance rather than other controls. The position coordinates and count of all the clickable elements are being tracked by the tracking agent. This task happens prior to page load. This feature makes it possible to obtain the exact information about the web page, before being loaded. The requested web page is loaded only after collecting all the necessary information. The collected information is stored in the local memory.

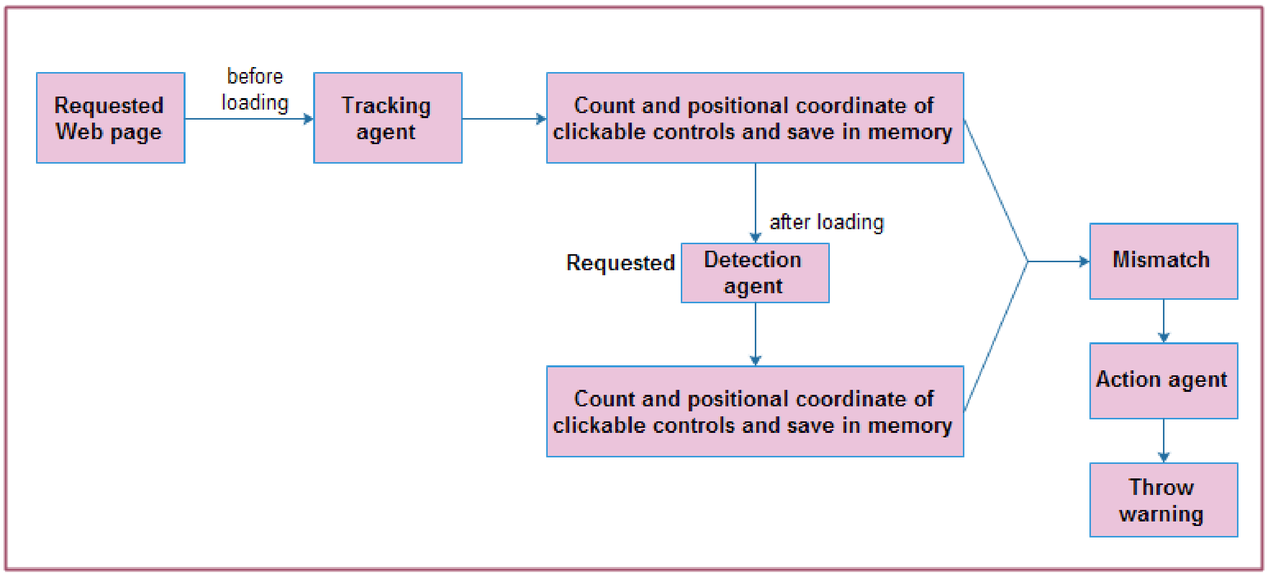
Requested Web page
before loading
Tracking agent
Count and positional coordinate of clickable controls and save in memory
after loading
Requested
Detection agent
Count and positional coordinate of clickable controls and save in memory
Mismatch
Action agent
Throw warning

Fig 2: Proposed work

B. *Detection agent*

Detecting agent starts to work after the loading of the page. In case, if a transparent webpage is superimposed by the adversary, the detection agent sniffs it by checking the count of clickable controls along with their positional coordinates. This can be accomplished by comparing the current webpage with the information stored in the database.

C. *Action agent*

The role of action agent begins when something goes wrong. In case, if any superimposition is detected by the detecting agent, then a warning is triggered as an alarm for the user.

D. *Overall flow of the proposed work*

1. Execute the page handler, whenever a new page is loaded.
2. The click handler routine, checks whether the control is clickable.
3. If the control is clickable, then the counter for control presence is incremented and the positional coordinates of the clickable control is saved in the local memory.
4. Load the webpage
5. Check the count and positional coordinates of the clickable elements of the webpage.
6. Compare it with the values stored in local memory
7. If the count of clickable controls is more than the count of clickable controls in the local memory, pass the control to action agent.
8. Action agent throws warning to the user.
9. Similarly, compare the positional coordinates of the loaded webpage with the positional coordinates of the local memory.
10. When all the registered listeners of the page get executed, stop triggering the mouse events.

Hence, the proposed work presents a scheme that eliminates clickjacking by exploiting three agents namely tracking, detection and action agents. Thus, the purpose of the system is attained.

IV. CONCLUSIONS

This paper presents a technique that withstands clickjacking attacks by means of agents. The agents being employed in this work are tracking, detection and action agents. The tracking agents are responsible for observing the web page controls and save the necessary information in the local memory. The detection agent is meant for comparison between the loaded webpage attributes with the parameters stored in local memory. In case of any mismatch occurrence, the action agent is triggered and it triggers the warning to the user.

REFERENCES

[1] Rahul Johari, Pankaj Sharma, "A Survey On Web Application Vulnerabilities (SQLIA,XSS)Exploitation and Security Engine for SQL Injection", International Conference on Communication Systems and Network Technologies, pp. 453-458, 2012.

[2] Isatou Hydara, Abu Bakar Md. Sultan, Hazura Zulzalil, Novia Admodisastro, "Current state of research on cross-site scripting (XSS) – A systematic literature review", Information and Software Technology, Vol. 58, pp.170–186, 2015.

[3] 4. Hansen, R.: Clickjacking, http://ha.ckers.org/blog/20080915/clickjacking/

[4] Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In: IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010) (2010)

[5] R. Hansen and J. Grossman. Clickjacking. http://www.sectheory.com/clickjacking.htm, 2008.

[6] L. C. Aun. Clickjacking with pointer-events. http:// jsbin.com/imuca.

[7] R. Hansen. Clickjacking details. http://hackers. org/blog/20081007/clickjacking-details/, 2008.

[8] E. Vela. About CSS Attacks. http:// sirdarckcat.blogspot.com/2008/10/about-css-attacks.html, 2008.

[9] Adobe. Flash OBJECT and EMBED tag attributes. http://kb2.adobe.com/cps/127/tn_12701.html, 2011.

[10] M. Zalewski. Minor browser UI nitpicking. http: //seclists.org/fulldisclosure/2010/Dec/328,2010.

[11] E. Vela. About CSS Attacks. http:// sirdarckcat.blogspot.com/2008/10/about-css-attacks.html, 2008.

[12] K. Kotowicz. Cursorjacking again. http://blog.kotowicz.net/2012/01/cursorjacking-again.html, 2012.

[13] E. Bordi. Proof of Concept - CursorJacking (noScript). http://static.vulnerability.fr/noscript-cursorjacking.html.

[14] M. Zalewski. Firefox focus stealing vulnerability (possibly other browsers). http://seclists.org/ fulldisclosure/2007/Feb/226, 2007.

[15] M. Zalewski. The curse of inverse strokejacking. http://lcamtuf.blogspot.com/2010/06/curse-of-inverse-strokejacking.html, 2010.

[16] J. Ruderman. Race conditions in security dialogs. http://www.squarefree.com/2004/07/01/ race-conditions-in-security-dialogs/, 2004.

[17] M. Zalewski. On designing UIs for non-robots. http://lcamtuf.blogspot.com/2010/08/on-designing-uis-for-non-robots.html, 2010.

[18] F. Aboukhadijeh. HOWTO: Spy on the Webcams of Your Website Visitors. http://www.feross.org/webcam- spy/, 2011.

[19] G. Aharonovsky. Malicious camera spying using ClickJacking. http://blog.guya.net/2008/ 10/07/malicious-camera-spying-using-clickjacking/, 2008.

[20] M. Zalewski. X-Frame-Options, or solving the wrong problem. http://lcamtuf.blogspot.com/2011/12/x-frame-options-or-solving-wrong.html, 2011.

[21] L.-S. Huang and C. Jackson. Clickjacking attacks unresolved. http://mayscript.com/blog/david/clickjacking-attacks-unresolved, 2011.

[22] J. Ruderman. Bug 162020 - pop up XPInstall/security dialog when user is about to click. https://bugzilla. mozilla.org/show_bug.cgi?id=162020, 2002.

[23] M. Mahemoff. Explaining the "Don't Click" Clickjacking Tweetbomb. http://softwareas.com/explaining-the-dont-click-clickjacking-tweetbomb, 2009.

[24] Wikipedia. Likejacking. http://en.wikipedia.org/ wiki/Clickjacking#Likejacking.

[25] S. Sclafani. Clickjacking & OAuth. http://stephensclafani.com/2009/05/04/ clickjacking-oauth/, 2009.

[26] E. Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849 (Informational), Apr. 2010.

[27] J. Grossman. Clickjacking: Web pages can see and hear you. http://jeremiahgrossman.blogspot.com/2008/10/clickjacking-web-pages-can-see-and hear.html, 2008.

[28] R. Hansen. Stealing mouse clicks for banner fraud. http://ha.ckers.org/blog/20070116/stealing-mouse-clicks-for-banner-fraud/, 2007.

[29] K. Kotowicz. Exploiting the unexploitable XSS with clickjacking. http://blog.kotowicz.net/2011/03/exploiting-unexploitable-xss-with.html, 2011.

[30] P. Stone. Next generation clickjacking. In Black Hat Europe,2010.