

CLICKSAFE: PROVIDING SECURITY AGAINST CLICKJACKING ATTACKS

Jawwad A. Shamsi^{#1}, Sufian Hameed^{#2}, Waleed Rahman^{#3},

Farooq Zuberi^{#4}, Kaiser Altaf^{#5}, Ammar Amjad^{#6}

[#]C.S Dept., National University of Computer and Emerging Sciences – Karachi
ST-4 Sector 17-D, on National Highway, Karachi, Pakistan
(021) 111-128-128

¹jawwad.shamsi@nu.edu.pk, ²sufian.hameed@nu.edu.pk, ³k092162@nu.edu.pk
⁴k092037@nu.edu.pk, ⁵k092065@nu.edu.pk, ⁶k092027@nu.edu.pk

Abstract— Clickjacking is an act of hijacking user clicks in order to perform undesired actions which are beneficial for the attacker. We propose Clicksafe, a browser-based tool to provide increased security and reliability against clickjacking attacks. Clicksafe is based on three major components. The detection unit detects malicious components in a webpage that redirect users to external links. The mitigation unit provides interception of user clicks and give educated warnings to users who can then choose to continue or not. Clicksafe also incorporate a feedback unit which records the user's actions, converts them into ratings and allows future interactions to be more informed. Clicksafe is predominant from other similar tools as the detection and mitigation is based on a comprehensive framework which utilizes detection of malicious web components and incorporating user feedback. We explain the mechanism of clicksafes, describes its performance, and highlights its potential in providing safety against click jacking to a large number of users.

Keywords— Clickjacking, Browser Security, Soft assurance of safe browsing, Security, Safety

I. INTRODUCTION

Assurance of safety from clickjacking is significant for web users. Ensuring this safety requires extensive mechanism of verification and validation against websites that hijack users' clicks in order to initiate a malicious action.

Clickjacking [1] is a very common attack through frames, in which, the user unknowingly clicks on a malicious page that sits on top a benign page. This is usually done by loading the malicious page as a transparent page over a benign page that genuinely requires a click or some input (Like user login, send email, etc). When the user gives the input that was asked, an event is sent to the malicious page (usually a click event) that causes some undesirable action to be taken on the user's behalf. Clickjacking is also referred to as UI redressing [3,4].

The most famous example of overlaying an invisible frame was the Clickjacking Tweet bomb [2] – in which a malicious page embedded Twitter.com on a transparent IFRAME. The victim page enticed the user by placing a 'Don't click' button directly above the invisible 'Tweet' button. If the user clicked on the button, a status message, which contained a link to the malicious website is posted on behalf of the user.

Clickjacking can also be implemented by simply hiding single UI elements, rather than the whole page in an IFRAME. Likejacking attacks [5] and Tapjacking attacks [6] are examples of such link of attacks. In all such attempts of attack, browser is the main source of attack. Therefore, our focus remains in the context of web browsers.

Many defences have been suggested for clickjacking for web browsers but they have all been circumvented by malicious users. Mostly, the defence consists of frame busting [7][8], which simply limits browser functionality by disallowing the IFRAME feature, but it does work as the webpage cannot get framed over another webpage. But the biggest problem with this technique is that it doesn't work with third-party widgets, such as Facebook 'Like' button. There are other defences that have been suggested (Discussed in Section II), but most have been circumvented, or more importantly can be circumvented later, and suffer from poor usability and incompatibility with webpages and widgets. According to Huang et al., clickjacking has not properly addressed and prevented [9]. It was found out that some test attacks still had an effectiveness of 98%.

During our literature review, we realized that Balduzzi [10] gave out the best mitigation results and was also the most cited and authoritative result. A major limitation of almost all of the existing techniques is that they try to work before the user has clicked the clickjacked link and offer no support for the false negatives that they can't detect. Consequently, if a prevention method is circumvented then it becomes essentially useless.

Considering the current defence techniques and some of their shortcomings, we kept the following objectives when developing our own defence to clickjacking:

- The solution should be compatible with old websites
- The solution should grow with time rather than remain stagnant.
- The solution shouldn't degrade the surfing speed.
- Even the most naive users should be able to make use of the solution.
- The solution should have the potential to be applied to other browser vulnerabilities as well.

To achieve the above goals, we developed our solution, called Clicksafe, which recognizes the potential malicious UI

elements and then upon a ‘click’ on these, informs the user of the intent of that particular click. For a naive user, Clicksafe will have an option of recommendations based on previous user’s experience; the user’s next action will then extend the ratings of this particular URL by choosing to go through with the click he made, or opting to stop the external redirection. We implemented a prototype of Clicksafe, as an add-on, on Mozilla Firefox Browser and found that it is very practical to use; depending upon the size of HTTP request, it added at most one second of delay for identifying a malicious click.

In summary, we make the following key contributions:

- We provide a through sum-up of the current detection techniques which try to mitigate clickjacking attacks. We also try to list down the shortcomings of tools like ClearClick technique introduced by NoScript [11].
- We present Clicksafe, a solution to clickjacking that we developed, which works by incorporating clickjacking detection and user feedback to ensure that even the most naive of users can make an educated guess, when clicking on a malicious link.

II. BACKGROUND AND RELATED WORK

Click jacking can be mitigated in a number of ways. It is imperative to note that almost all of the techniques listed below, work before the actual click is done. Hence they offer no support if a false negative is made. Existing techniques can be classified into 3 categories based on how the mitigation is being done:

- Browser Add-on/Browser based: It refers to mitigation techniques that are solely based upon the browser or an add-on. In such a case, the browser itself is responsible for the mitigation of clickjacking. Examples of such add-ons are NoScript [11] and FlashBlock [12].
- Website code/script: These mitigation techniques are those that are implemented on the website where the website is solely responsible for mitigation. Examples of such techniques include frame busting scripts [7], [8] which disable iframes from employing clickjacking.
- Website + Browser Code: This category represents a new, yet more difficult to implement techniques which require coordination from the browser and the website. The first of these techniques was the X-Frame options [13] presented by IE8 and soon adopted by other browsers as well. This category of techniques requires that the browser is capable of utilizing the technique and that the website also employs the relevant code.

A. Detection Techniques

1) Browser Add-on/Browser based:

- Disable onBeforeUnload event to ensure frame busting: In this technique, a user can manually cancel any navigation request submitted by a framed page. Whenever a framing page is about to be unloaded due to navigation, an onBeforeUnload handler is called.
- ClearClick: This is an extension to the No script add-on that especially catered to clickjacking. Whenever a user

interacts with an embedded element which is partially obstructed, transparent or otherwise disguised, clearclick [11] will intercept the action and reveal the hidden elements. It supports desktop and mobile versions via the NoScript add-on. It also works like Clicksafe in that it aims to educate the naive user.

- FlashBlock Firefox Add-on: Flashblock [12] is an extension for the Mozilla, Firefox, and Netscape browsers that block all Macromedia Flash content on a webpage from loading. It is an extreme approach that limits features in order to prevent clickjacking.
 - Opaque Overlay Policy: The Gazelle web browser [14] adopted a method that forced all cross-origin frames to be shown opaquely. However, this approach causes many benign sites to break down.
 - Disable windows switching via JavaScript Code: JavaScript allows the frames and windows to be loaded on to another webpage, scripts of these sort are disabled in order to provide security regarding clickjacking.
 - CSS checking (using position, size, opacity and z-index): The page is parsed to check any overlapping and invisible elements based on CSS characteristics. For instance, in order to block mouse clicks a browser may detect that the clicked cross-origin frame is not fully visible. Applications like Adobe have added such protection to protect against webcam being forcibly compromised.
 - Disable all JavaScript: This works in practice just like Flash block, i.e., it blocks all JavaScript [15] from the page and greatly limits the functionality of the webpage and user experience rather than address clickjacking as a whole.
 - Freezing of DOM to check on elements: Using new features on ECMA Script 5th Edition, this works by freezing objects so their properties become unchangeable [19]. This prevents malicious code from changing object properties in a potentially malicious manner inducing clickjacking.
- #### 2) Website code/script:
- Using the Same Origin Policy on a Website to avoid attacks: The same origin policy [18] prevents a webpage from loading a document or script from one origin in order to get or set properties of a document from a different origin.
 - Frame Busting of iframes: Since most of the clickjacking that was being done, was via iframes. Disallowing iframes, on which a malicious element could be loaded, all together also becomes a very efficient technique. This can be done with JavaScript code in the potentially targetable element that ensures it stays on top [8]. But this is also easily overcome by attackers as Zalewski [17] and Huang [9] showed how to bypass framebusting.
 - UI Randomization: UI randomization works by making it harder for the attacker by randomizing the concerned, usually clickable, element’s location/UI layout [16].

This is not very efficient as the attacker can still guess the location or ask the victim client to perform multiple clicks that correspond to guess clicks.

3) Website + Browser Code:

- HTTP X-Frame Options to enforce Same Origin Policy: This works in exactly the same way as framebusting but with added browser support, using features called X-Frame-Options [21]. Also it has the same limitations as that of frame busting.
- Static Reference Bitmap: This method basically entails taking and comparing the picture/screenshot [9] of the region of the clickable element, and the bitmap of the same clickable element rendered at the time when its shown. If these two bitmaps are not the same, then the click action of the user is cancelled and redirection is stopped. But a limitation stays that the website itself has to provide the bitmap.
- Disable cursor customization on sensitive elements: Cross-Origin cursor customization is not permitted to ensure that the user does not make a false click, because a fake cursor provides false pointer location to the user.
- Screen Freezing on sensitive elements: A naive user will usually try to click a animated object and thus be redirected to a malicious link. To counter this, rendering updates are stopped, i.e. frozen so that the user can make a correct click.
- Lightbox/Greyout on sensitive elements: The intensity of the region around the area of the click is darkened every time a frame is opened that allows external redirection. This approach is used by Facebook
- InContext Defense System: InContext [9] makes sure that the UI element is not presented out of context to the user by marking all elements that are sensitive and asking browsers to enforce the timing and context of the user.

B. Discussions

As we mentioned earlier, all of these defences take place before the actual click has taken place and offer no support for if the user has already clicked on the malicious link.. In Section 3, we will introduce Clicksafe - a browser Add-on that tries to reduce false positives via user feedback so that even if the detection technique gets circumvented, the naive user can still make an educated guess.

Whereas most existing defences have provided a solution to mitigate clickjacking before the click has taken place, in Clicksafe, a user can go back and change her feedback for a webpage so that she doesn't repeat the mistake. We also ensure the backward compatibility of our solution in that it doesn't break a webpage's code and works for every webpage, making an add-on that is also very practical to use and provides better feedback with increasing number of users.

III. CLICKSAFE: CLICKJACKING DETECTION AND MITIGATION APPROACH

In this section we will present our approach to mitigate clickjacking via user community feedback; we will also illustrate the consequences of the user chancing upon a previously flagged clickjacking website (by user community). Once the DOM has finished loading, we parse the page for malicious redirect intent and possible clickable elements (that is, elements which the user can interact with). After attaching scripts to the loaded page and modifying its code, our add-on will intercept user clicks by displaying a popup and stopping event execution whenever the user clicks on a clickable element.

Fig. 1 illustrates the architecture of our system. It consists of three main parts: a detection unit in which the add-on searches for any possible insecure elements that contain implicit redirection code, a mitigation unit in which we make the user aware of the potential threat of her click and a feedback unit which allows the user to give feedback to the add-on on the validity of the suggestion shown.

A. Detection Unit:

This unit is responsible for detecting clickjacking on a rendered webpage by searching for an element that is redirecting the user to an external page both implicitly via JavaScript as well as explicitly via HTML anchor tags. Implicit redirection detection is carried out when a page loads. Clicksafe will first pull all external and internal JavaScript from the page, analyze it using a parser, and then recursively check for redirection. JavaScript can cause a page to redirect in multiple ways, some of which are as follows:

```
window.location.href="";
window.navigate();
self.location="";
```

By searching for instances of such pieces of code recursively inside a parsed JavaScript tree, we were able to locate if a specific link was implicitly redirecting the user to another page or not. As for explicit redirection attempts, the following functions are used to prevent the anchor tag event handler from the executing the hyperlink.

```
preventDefault();
stopPropagation();
```

The hyperlink itself is later executed, when the user confirms that she wants to be redirected a specific page.

B. Mitigation Unit:

Clickjacking mitigation mainly works by intercepting user clicks and giving educated warnings to users who can then choose to continue or not. When an external link is clicked, the click is intercepted and a popup displaying both the URL she clicked on, as well as the site's rating is presented to the user. The user then has the option to either continue with the action or cancel her redirect attempt. As for redirection, if a DOM element has redirection code, its default onClick event handler will be replaced with another that executes a popup displaying a warning. Upon the user's confirmation, the original JavaScript code will proceed normally.

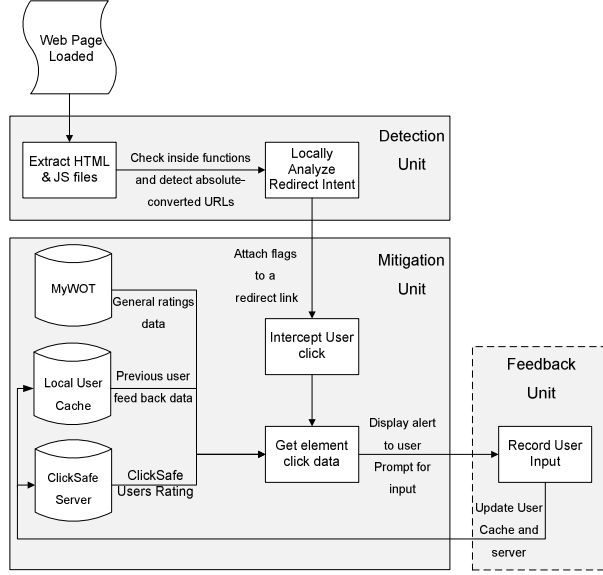


Fig. 1 Clicksafe System Architecture

On positive detection of a threatening element, the add-on can inform the user via user community rating or by illustrating the user explicitly that there are in fact two elements she is clicking on. If user ratings are available for the element, the add-on will highlight the element that majority of the user previously clicked on and also show a rating/percentage of the community that agrees with it. Thus, informing the user that indeed she has been saved from a clickjacking attempt.

Alternately, if the add-on does not have previous rating from the user, then it will highlight both the overlapping elements and present the user with an option to select an element that she thinks is appropriate.

C. Feedback Unit:

The most crucial part of the system is the feedback unit, which records the user's actions, converts them into ratings and allows future interactions to be more informed.

When a popup intercepting a link appears, it displays a number of ratings. Firstly, it has an external rating by MyWOT which displays a range of the score the domain has. This external rating is used for a number of reasons. Firstly, ClickSafe does not have a pre-built list for blacklisted and whitelisted domains; initially all websites will have a rating of 0. Building such a list is infeasible unless a crawler is generating a list constantly. So, initially, ClickSafe will have to rely on external information. Secondly, the MyWOT rating also provides an extra piece of information to judge whether the requested link is trustworthy or malicious.

The second piece of feedback is of the ClickSafe community itself. Every time the user is presented with a popup and responds by clicking a yes or a no, her actions are recorded and submitted to a server. To the question, "Do you want to continue?" preceded by information about the link, a yes will give the site a positive rating while a no will give the

site a negative rating. This feedback can be changed later if the user has accidentally provided the wrong feedback. Feedback from the ClickSafe community is aggregated if it meets a certain confidence level and displayed to the user as a percentage of positive vs. negative votes.

Finally, the user's own click is essential in determining feedback and increasing usability. If the user has negatively rated a page, the next visit to the page will be marked with a notification bar at the top of the page which informs the user about her own previous rating to the site.

D. Limitations

Some of the limitations of the system can be traced to the fragility of user trust and the unreliability of the user's response. User feedback is generally treated to be unreliable [20]. Even if all users provide accurate feedback, malicious users can exploit the system and give malicious, incorrect feedback. Further, bots using unique user IDs can also disrupt ratings of websites.

Another limitation of our scheme is related to click interception. In terms of the effectiveness of our solution, clicks on webpages are captured through four different methods: via traditional HTML anchor tags, using flash-based or other external dynamic objects, through JavaScript event handlers and JavaScript event listeners. Our solution has been designed to intercept clicks of the first two methods though our redirection code focused on event handlers. JavaScript event listeners, which allow multiple events to be attached to an event handler, are not covered.

A third limitation is the dynamic nature of JavaScript itself. JavaScript code can start out as initially as a mere fifty lines of code and then multiply itself via Ajax calls and function rewriting to produce hundreds of lines of code. This method is utilized quite extensively such as in case of Google jQuery. This disrupts static analysis of code and dynamic analysis techniques will be required which are much of the time slower than static preprocessing [21]. Finally, redirection scripts used by websites further alleviate the problem. Google, for its part, links to a local URL which then redirects to the target URL. This roundtrip is a serious problem for our add-on which cannot identify the target URL beforehand.

IV. RESULTS

Table 2 shows the difference in page load time with Clicksafe in comparison to another Firefox add-on, LORI (Life of Request Info). For some small sites, it took only a few milliseconds while for others it took seconds. This difference in load time can be attributed to different factors, though the most prominent is the page size itself. Fig. 2 shows the linear relationship between page size and page load time. Part of the reason for this is that our current implementation iterates through each element checking it is clickable and then modifies the page's code accordingly. If this preprocessing could be incorporated while the page is being loaded or while the user is busy with other actions, it may reduce the load time. As explained in the previous section, our add-on was unable to detect all clickable elements on websites because of the following factors:

Table 1 – Webpage load time

Load Time (sec)		Size (KB)	Time Diff (sec)
w/o Clicksafe	With Clicksafe		
3.961	4.248	610.5	0.287
0.420	0.47	16	0.05
1.73	1.81	11	0.08
3.592	5.338	1330	1.74
7.090	8.368	1650	1.27
1.084	2.366	515	1.3
7.3	10.906	483	3.60

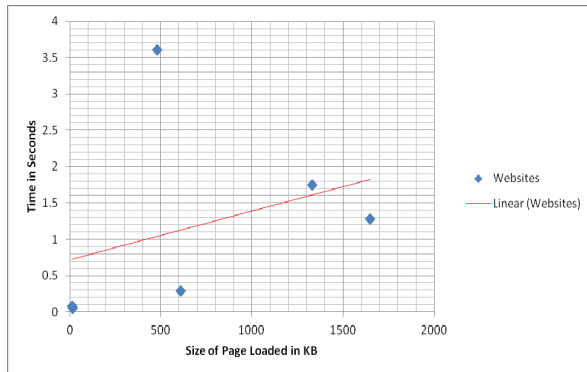


Fig. 2 - Page Load time as a function of page size

- JavaScript Obfuscation
- Dynamic processing and rewriting of JavaScript code
- Event Listeners
- URL redirection

Event listeners, though not covered currently, can be easily tracked and parsed by modifying object prototypes. URL redirection is more difficult to detect as the only way to check the end URL is by traversing the path itself. JavaScript obfuscation is also difficult to undo, though it can be reversed partially via an efficient parser. Dynamic processing, on the other hand, can only be mitigated by dynamic analysis.

V. CONCLUSION AND FUTURE WORK

This paper presents a novel approach to counter clickjacking. The solution utilizes user feedback to create dynamic black and white lists and overcome limitations posed by previous solutions. Despite a few limitations, Clicksafe is effective in providing security against clickjacking attacks.

We plan to extend this work by building dynamic analysis of JavaScript code obfuscation methods as well as an encryption/authentication system which does not allow the voting system to be compromised. Though Firefox add-ons are open-source, and the client-side code can be easily modified for malicious intent, secure protocols which allow for secure transmission can be considered. Finally, considering the fact

that our system is similar to a recommender system, trust of each user can be assessed and evaluated using trust-based metrics.

REFERENCES

- [1] R. Hansen. Clickjacking. ha.ckers.org/blog/20080915/clickjacking. Last accessed July 31st, 2013
- [2] M. Mahemoff. Explaining the “Don’t Click” Clickjacking Tweetbomb. <http://softwareas.com/explaining-the-dont-click-clickjacking-tweetbomb,2> 2009.
- [3] M. Zalewski. Browser security handbook. [http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_\(UI_redressing\)](http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_(UI_redressing)).
- [4] M. Niemi. “UI Redressing: Attacks and Countermeasures Revisited”. In CONFIDENCE, 2011.
- [5] R. Lunde, O. Jesse, R. Travis. “New Ways I’m Going to Hack Your Web App”. Blackhat AD, 2011.
- [6] G. Rydstedt, B. Gourdin, E. Bursztein, D. Boneh, “Framing Attacks on Smart Phones and Dumb Routers: Tap-jacking and Geo-localization Attacks”. Proceedings of the 4th USENIX conference on Offensive technologies. USENIX Association, 2010.
- [7] E. Lawrence. IE8 Security Part VII: ClickJacking Defenses. <http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx>, 2009.
- [8] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. “Busting frame busting: a study of clickjacking vulnerabilities” at popular sites. In Proceedings of the Web 2.0 Security and Privacy, 2010.
- [9] Huang, L. S., Moshchuk, A., Wang, H. J., Schechter, S., & Jackson, C. “Clickjacking: Attacks and Defenses”, 2012. Usenix Security Symposium, 2012.
- [10] M. Balduzzi, M. Egele, E. Kirda, D. Balzarotti, and C. Kruegel. “A solution for the automated detection of clickjacking attacks.” In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 135-144. ACM, 2010.
- [11] Maone, G. Noscript clearclick. <http://noscript.net/faq#clearclick>, January 2012.
- [12] Philip Chee. FlashBlock Browser Add-on: <http://flashblock.mozdev.org/faq.html>
- [13] D. Ross, T. Gondrom, Thames Stanley. “HTTP Header Field X-Frame-Options” Draft for WEBSEC IETF, 2013.
- [14] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter. The Multi-Principal OS Construction of the Gazelle Web Browser. In Proceedings of the 18th Conference on USENIX Security Symposium, 2009.
- [15] Maone, G. Noscript. <http://noscript.net/faq>, January 2012.
- [16] B. Hill. Adaptive user interface randomization as an anti-clickjacking strategy. http://www.thesecuritypractice.com/the_security_practice/papers/AdaptiveUserInterfaceRandomization.pdf, May 2012.
- [17] M. Zalewski. X-Frame-Options, or solving the wrong problem. <http://lcamtuf.blogspot.com/2011/12/x-frame-options-or-solving-wrong.html>, 2011.
- [18] J. Ruderman. The Same Origin Policy. <http://www.mozilla.org/projects/security/components/same-origin.html>, 2011.
- [19] M. Heiderich, F. Tilman, and H. Thorsten. “Iceshield: Detection and mitigation of malicious websites with a frozen dom.” In Recent Advances in Intrusion Detection, pp. 281-300. Springer Berlin Heidelberg, 2011.
- [20] J. Moore, and C. Paris. “Exploiting user feedback to compensate for the unreliability of user models.” User Modeling and User-Adapted Interaction 2, no. 4 (1992): 287-330.
- [21] Ernst, Michael D. “Static and dynamic analysis: Synergy and duality.” In WODA 2003: ICSE Workshop on Dynamic Analysis, pp. 24-27. 2003.