Draft

ClearClick: Effective Client-Side Protection Against UI Redressing Attacks

Giorgio Maone <giorgio at maone.net>

Rev. 2, May 3, 2012

Abstract

"User Interface Redressing", popularized in 2008 as "Clickjacking", designates a class of attacks, leveraging ambient authority and the coexistence in modern user agents of multiple browsing contexts, which trick an authorized human into interacting with UI elements that actually belong to the targeted web application, but have been obscured or decontextualized by attacker-provided content. This interaction induces unintended application state changes on behalf of the victim – similarly to Cross Site Request Forgery – but defeats traditional CSRF protections, such as form tokens, by exploiting the legitimate web application UI itself, which those countermeasures are meant to validate. The main defense currently adopted by mainstream browsers requires a server-side opt-in signal and prohibits legitimate, widespread use cases such as cross-site subdocument embedding. Another countermeasure immune to these limitations (enabled by default, entirely client-side and designed to allow cross-domain embedding) is the ClearClick module, included in the NoScript add-on for Mozilla Firefox just days after the first Clickjacking announcement. This document describes the rationale behind it and the way it works.

Keywords: clickjacking, anti-clickjacking, UI redressing, ClearClick, X-Frame-Options.

1. Introduction

In September 2008, Jeremiah Grossman and Robert "RSnake" Hansen canceled a previously announced OWASP talk as requested by Adobe[1], whose Flash Player plugin was said vulnerable to a mysterious new attack called "Clickjacking"[2]. As they explained later, this attack consisted in a web page embedding an invisible but clickable instance of the "Flash Settings" applet (made transparent via the opacity CSS property), which was set to follow the mouse pointer by a trivial piece of JavaScript code: wherever on the page the user clicked, the click was actually received by the checkbox which enabled microphone capture, allowing the attacker to perform "web-based wiretapping", as the press liked to sensationally report at that time. Even though Adobe fixed the specific Flash vulnerability, it was pretty clear that this was just an example of a much broader and difficult to fix issue with modern web browsers: the coexistence of multiple browsing contexts in the same user agent UI (tabs, windows) or even in the same page (frames, iframes, plugin content) could be used to bypass the Same Origin Policy and traditional Cross-site Request Forgery protections by exploiting a misperception of the user interface's identity and directing user's input to a web application different than the one he or she intended to interact with. This problem had been already somehow identified, but its security implication went ignored or misunderstood and underestimated for a long time, as demonstrated by the discussion on a Mozilla bug report by Jesse Ruderman in 2002.[3] The drama around the Clickjacking "forbidden talk" raised the awareness in the web security community and browser vendors slowly started thinking about possible countermeasures.[4] [5] [6]

2. ClearClick's Roots and Rationale

Before Clickjacking's details were revealed by Grossman & Hansen, the term “UI Redress(ing)” had been introduced by Michal Zalewski in a seminal post on the WHATWG mailing list[7], where he acknowledged the problem as inherent to current browser technology and, more importantly, drafted some possible defenses to be implemented by user agents, which had emerged from private discussions involving him and a number of other researchers at Google. Of those five proposals, only the first one found its way almost verbatim in mainstream browser technology, despite its several shortcoming, probably because, in Zalewski's own words, it is “super-simple”: the "X-I-Do-Not-Want-To-Be-Framed-Across-Domains: yes" HTTP header, which was going to be adopted some months later by Microsoft's Internet Explorer 8 as “X-Frame-Options” and is supported by all the major web browsers nowadays. Unfortunately, the objections Zalewski himself had outlined about this countermeasure are still mostly valid:

- *"Opt-in", i.e. currently vulnerable sites remain vulnerable unless action is taken*
- *Can't be used for cases where IFRAME content mixing has a legitimate purpose (for example, cross-domain gadgets, certain types of mashups)*
- *Adds yet another security measure (along with cross-domain XHR, MSIE8 XSS filters, MSIE P3P cookie behavior, Mozilla security policies) that needs to be employed correctly everywhere to work - which is very unlikely to consistently happen in practice*
- *Along with the aforementioned security features, threatens to result in HTTP header or HTML HTTP-EQUIV size bloat that some sites may care about*[8]

The other proposals have been variously discussed over the years, but never implemented in practice. The most interesting, very detailed and endorsed by Zalewski himself, apparently because it “works by default” and overcomes the limitations above, was the number 3:

> *Add an on-by-default mechanism that prevents UI actions to be taken when a document tries to obstruct portions of a non-same-origin frame. By carefully designing the mechanism, we can prevent legitimate uses (such as dynamic menus that overlap with advertisements, gadgets, etc) from being affected, yet achieve a high reliability in stopping attacks.*
>
> *[I like this one the most myself, but we were far from reaching any consensus.]*
>
> *Algorithm description: [...]*[9]

Incidentally, the rationale behind ClearClick, the anti-clickjacking module included in the NoScript add-on for the Mozilla Firefox browser less than a couple weeks later[10], is pretty much the same. ClearClick's algorithm design and implementation is significantly different, though, because rather than leveraging ad hoc callbacks from the renderer and low-level input disablement, which would have required customizing the browser's core code and possibly incurring in performance penalties, it makes a creative use of readily available HTML 5 technologies such as the canvas element and DOM event capturing, plus a few extra privileges and APIs made available by the Mozilla platform for extensions.

3. ClearClick's Algorithm

ClearClick, as a module of the NoScript add-on for the Mozilla Firefox browser, is granted chrome privileges (the same as the browser code) and can use a number of internal Mozilla APIs which are not available to unprivileged web content nor found in other browsers. Notwithstanding, most of its code (which is entirely written in JavaScript) leverages technologies, such as the HTML 5 canvas element, DOM events or DOM manipulation, which belong to the web platform. In the following description we will emphasize the bits which are not easily portable, because require privileges and/or APIs unavailable to standard web content, within a *CBC* (Cross-Browser Caveat) section.

Algorithm:

1. **Listener registration** - Register a “global” capturing event listener for mouse button, tapping, keyboard, drag & drop and focus events, which must be guaranteed to run before any other event handler of the same kind and therefore be able to prevent any event from being handled by the content, if needed.
   CBC: in order to guarantee the “first to process” event listener requirement and reduce registration overhead, ClearClick adds its listener to the Mozilla-specific DocShell object which is the immediate container of the topmost DOM window per any given tab. A cross-browser approach likely to work is registering the listener on the topmost DOM window itself before any script has a chance to run.
2. **Fast-track bypass** - Whenever the listener is called, check whether the event target or its owner document are flagged as “unlocked”. If either is, return early.
   CBC: ClearClick uses an expando property to flag DOM nodes and windows, relying on a feature of Mozilla's chrome-exposed DOM wrappers which prevents content from seeing or tamper with expando properties set by privileged code. Other browsers may require different procedures to safely annotate documents and other DOM nodes. Furthermore, this and most of the remaining steps assume our listener can examine and manipulate any DOM node or window independently from its origin, bypassing SOP. This privilege should be granted by the listener having being registered by privileged (browser extension) code.
3. **Parent chain check** – Check whether the event target is either a child of a nested document or a plugin content element (EMBED, APPLET or OBJECT). If it is not, or it is an embedded document belonging to a same-site parent chain (i.e. it and all its parents are from the same origin), flag the document as “unlocked” and return. Notice that the original Clickjacking demo by Hansen & Grossman worked despite the Flash content being served same-site: since plugins may follow type-specific origin policies, we never return early at this stage when interacting with plugin content, even if embedded same-site.
4. **Rapid fire check** – Check whether the previous event we had observed was the same type on a document from a different origin, happened within the past 800ms (quarantine time). If it was, we assume a “Rapid fire” attack (e.g. the user has been tricked into repeatedly click on the same or a predictable location in a fast succession while the document gets changed under his mouse pointer): halve the quarantine time and go to step 8. If next interaction happens with a different document, the quarantine time will be reset.
5. **Cursor sanity check** – By querying computed-style with the “:hover” pseudo-class on the element (if the target is plugin content) or on the host frame element and its ancestors (if the target is a nested document), check whether the cursor has been hidden or changed to an possibly attacker-provided bitmap: if it has, go to step 7. This provides protection against “Phantom cursor” attacks, also known as “Cursorjacking”.
6. **Obstruction check –** By using an offscreen HTML 5 canvas element, we take two reasonably sized (300x200 on average, but growing or shrinking depending on document's inherent size and viewport constraints) screenshots of the region centered around the DOM element which is about to receive the event: one from its owner document's “point of view” (unobstructed by definition), the other from the topmost window's. In the plugin content case, we ensure the former “screenshot” contains the element itself only. If the number of the pixels which are different between the screenshots don't exceed a certain configurable tolerance rate (default 18%), return. Otherwise we tentatively assume the DOM element our user is interacting with has been obstructed or obscured by a UI Redressing attempt.
   CBC: the screenshots are taken by using the CanvasRenderingContext2D.drawWindow() method[11], which is a Mozilla-proprietary extension of the HTML 5 Canvas API available to privileged code only, allowing the content of DOM windows to be drawn on a canvas surface exactly as rendered on the screen. The rest of this phase relies on cross-browser canvas features, instead, such as pixel grabbing and data URL serialization.

7. **User notification** - spawn a dialog which shows both the screenshots for easy visual comparison and detection of false positives.
8. **Interaction cancellation** - prevent the event from being processed, by using the ordinary DOMEvent API, then log the suspicious activity to the Error Console and return.

4. Future Directions

Some experimental work is being done with the *MozAfterPaint* Mozilla-proprietary event to explore the feasibility of an algorithm which sorts currently visible regions by origin and appearance time, in order to handle more efficiently timing-based attacks.

5. Conclusions

The ClearClick module included in the NoScript add-on for the Mozilla Firefox browser is currently the most effective client-side protection against various forms of UI Redressing attacks. It is enabled by default (independently from web author's opt-in), protects plugin content as well as embedded documents and doesn't impose origin restrictions on the nesting hierarchy. Unfortunately its main issue is the relative complexity of its implementation, which depends on a few Mozilla-specific platform features, even though it's entirely written in JavaScript and mostly relies on portable HTML 5 features. Here we offered a short, high-level description of its inner workings, putting emphasis on the less portable bits, with the hope of facilitating enhancement and porting efforts or stimulating ideas for new cross-browser solutions which overcome *Frame-Option*'s limitations.

1 Adobe PSIRT, *Thanks to Jeremiah Grossman and Robert "RSnake" Hansen*, Sep 2008, http://blogs.adobe.com/psirt/2008/09/thanks_to_jeremiah_grossman_an.html
2 Robert Hansen, Jeremiah Grossman, *Clickjacking*, Dec 2008, http://www.sectheory.com/clickjacking.htm
3 Jesse Ruderman, *Bug 154957: iframe content background defaults to transparent*, Jun 2002 https://bugzilla.mozilla.org/show_bug.cgi?id=154957
4 Eric Lawrence (Microsoft), *IE 8 Security Part VII: Clickjacking Defenses*, Jan 2009, http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx
5 Adam Barth (Google), *Security in Depth: New Security Features*, Jan 2010, http://blog.chromium.org/2010/01/security-in-depth-new-security-features.html
6 Michael Coates (Mozilla), *X-Frame-Options*, Sep 2010, http://blog.mozilla.org/security/2010/09/08/x-frame-options/
7 Michal Zalewski, *Dealing with UI redress vulnerabilities inherent to the current web*, Sep 2008, http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2008-September/016284.html
8 Ibid.
9 Ibid.
10 Giorgio Maone, *Hello ClearClick, Goodbye Clickjacking*, Oct 2008, http://hackademix.net/2008/10/08/hello-clearclick-goodbye-clickjacking/
11 Mozilla, *CanvasRenderingContext2D.drawWindow()*, https://developer.mozilla.org/en/DOM/CanvasRenderingContext2D#drawWindow%28%29