Clickjacking

Clickjacking (**User Interface redress attack**, **UI redress attack**, **UI redressing**) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages.[1][2][3][4] It is a browser security issue that is a vulnerability across a variety of browsers and platforms. A clickjack takes the form of embedded code or a script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function.[5] The term "clickjacking" was coined by Jeremiah Grossman and Robert Hansen in 2008.[6][7] Clickjacking is as an instance of the confused deputy problem, a term used to describe when a computer is innocently fooled into misusing its authority.[8]

Description

Clickjacking is possible because seemingly harmless features of HTML web pages can be employed to perform unexpected actions.

A clickjacked page tricks a user into performing undesired actions by clicking on a concealed link. On a clickjacked page, the attackers load another page over it in a transparent layer. The users think that they are clicking visible buttons, while they are actually performing actions on the invisible page. The hidden page may be an authentic page; therefore, the attackers can trick users into performing actions which the users never intended. There is no way of tracing such actions to the attackers later, as the users would have been genuinely authenticated on the hidden page.

Examples

A user might receive an email with a link to a video about a news item, but another webpage, say a product page on Amazon.com, can be "hidden" on top or underneath the "PLAY" button of the news video. The user tries to "play" the video but actually "buys" the product from Amazon. The hacker

can only send a single click, so they rely on the fact that the visitor is both logged into Amazon.com and has 1-click ordering enabled.

Other known exploits include

- Tricking users into enabling their webcam and microphone through Flash (though this has been fixed since originally reported)[9]
- Tricking users into making their social networking profile information public[*citation needed*]
- Downloading and running a malware (malicious software) allowing to a remote attacker to take control of others computers[10][11][12]
- Making users follow someone on Twitter[13]
- Sharing or liking links on Facebook[14][15]
- Getting likes on Facebook fan page[16] or +1 on Google Plus
- Clicking Google Adsense ads to generate pay per click revenue[17]
- Playing YouTube videos to gain views
- Following someone on Facebook

While technical implementation of these attacks may be challenging due to cross-browser incompatibilities, a number of tools such as BeEF or Metasploit Project offer almost fully automated exploitation of clients on vulnerable websites. Clickjacking may be facilitated by - or may facilitate - other web attacks, such as XSS.[18][19]

Likejacking

Likejacking is a malicious technique of tricking users of a website into "liking" a Facebook page that they did not intentionally mean to "like".[20] The term "likejacking" came from a comment posted by Corey Ballou in the article *How to "Like" Anything on the Web (Safely)*,[21] which is one of the first documented postings explaining the possibility of malicious activity regarding Facebook's "like" button.[22]

According to an article in *IEEE Spectrum*, a solution to likejacking was developed at one of Facebook's hackathons.[23] A "Like" bookmarklet is available that avoids the possibility of likejacking present in the Facebook like button.[24]

Cursorjacking

Cursorjacking is a UI redressing technique to change the cursor from the location the user perceives, discovered in 2010 by Eddy Bordi, a researcher at Vulnerability.fr,[*citation needed*] Marcus Niemietz demonstrated

this with a custom cursor icon, and in 2012 Mario Heiderich by hiding the cursor.[25][26]

Jordi Chancel, a researcher at Alternativ-Testing.fr, discovered a cursorjacking vulnerability using Flash, HTML and JavaScript code in Mozilla Firefox on Mac OS X systems (fixed in Firefox 30.0) which can lead to arbitrary code execution and webcam spying.[27]

A second CursorJacking vulnerability was again discovered by Jordi Chancel in Mozilla Firefox on Mac OS X systems (fixed in Firefox 37.0) using once again Flash, HTML and JavaScript code which can lead also to the spying of the webcam and the execution of a malicious addon allowing the execution of a malware on the computer of the trapped user.[28]

Password manager attack

A 2014 paper from researcher at the Carnegie Mellon University found that whilst browsers refuse to autofill if the protocol on the current login page is different from the protocol at the time the password was saved, some password managers would insecurely fill in passwords for the http version of https-saved passwords. Most managers did not protect against iFrame and redirection based attacks and exposed additional passwords where password synchronization had been used between multiple devices.[29]

Prevention

Client-side

NoScript

Protection against clickjacking (including likejacking) can be added to Mozilla Firefox desktop and mobile[30] versions by installing the NoScript add-on: its ClearClick feature, released on 8 October 2008, prevents users from clicking on invisible or "redressed" page elements of embedded documents or applets.[31] According to Google's "Browser Security Handbook" from year 2008, NoScript's ClearClick is "the only freely available product that offers a reasonable degree of protection" against Clickjacking.[32] Protection from the newer cursorjacking attack was added to NoScript 2.2.8 RC1.[25]

GuardedID

GuardedID (a commercial product) includes client-side clickjack protection for users of Internet Explorer and Firefox[33] without interfering with the operation of legitimate iFrames. GuardedID clickjack protection forces all frames to become visible.

Gazelle

Gazelle is a Microsoft Research project secure web browser based on IE, that uses an OS-like security model, and has its own limited defenses against clickjacking.[34] In Gazelle, a window of different origin may only draw dynamic content over another window's screen space if the content it draws is opaque.

Server-side

Framekiller

Web site owners can protect their users against UI redressing (frame based clickjacking) on the server side by including a framekiller JavaScript snippet in those pages they do not want to be included inside frames from different sources.[32]

Such JavaScript-based protection, unfortunately, is not always reliable. This is especially true on Internet Explorer,[32] where this kind of countermeasure can be circumvented "by design" by including the targeted page inside an <IFRAME SECURITY=restricted> element.[35]

X-Frame-Options

Introduced in 2009 in Internet Explorer 8 was a new HTTP header `X-Frame-Options` which offered a partial protection against clickjacking[36] [37] and was shortly after adopted by other browsers (Safari,[38] Firefox,[39] Chrome,[40] and Opera[41]). The header, when set by website owner, declares its preferred framing policy: values of `DENY`, `SAMEORIGIN`, or `ALLOW-FROM` *`origin`* will prevent any framing, framing by external sites, or allow framing only by the specified site, respectively. In addition to that, some advertising sites return a non-standard `ALLOWALL` value with the intention to allow framing their content on any page (equivalent of not setting X-Frame-Options at all).

In 2013 the X-Frame-Options header has been officially published as RFC 7034,[42] but is not an internet standard. The document is provided for informational purposes only.

Content Security Policy

The `frame-ancestors` directive of Content Security Policy (introduced in version 1.1) can allow or disallow embedding of content by potentially hostile pages using iframe, object, etc. This directive obsoletes the X-Frame-Options directive. If a page is served with both headers, the frame-ancestors policy should be preferred by the browser.[43]—although some popular browsers disobey this requirement.[44]

Example frame-ancestors policies:

```
# Disallow embedding. All iframes etc. will be blank, or contain a
browser specific error page.
Content-Security-Policy: frame-ancestors 'none'

# Allow embedding of own content only.
Content-Security-Policy: frame-ancestors 'self'

# Allow specific origins to embed this content
Content-Security-Policy: frame-ancestors www.example.com
www.wikipedia.org
```