White Paper

context
INFORMATION SECURITY LTD

Next Generation Clickjacking

New attacks against framed web pages

Paul Stone
(stone@contextis.co.uk)

14th April 2010

Abstract

Clickjacking is a term first introduced by Jeremiah Grossman and Robert Hansen in 2008 to describe a technique whereby cross-domain attacks are performed by 'hijacking' user-initiated mouse clicks to perform actions that the user did not intend[1]. In this paper, I will explore other ways a user can be tricked into interacting with a framed web page, that could allow an attacker to inject arbitrary text into forms and extract content from a web page. I will also show a new technique that allows information leaked from an iframe to be used for login detection and many other purposes.

Introduction

The clickjacking technique was introduced in 2008 by Robert Hansen and Jeremiah Grossman as a way to perform cross-domain attacks by ‘hijacking’ user-initiated mouse clicks to perform actions that the user did not intend[1]. To achieve this, an attacker will choose a clickable region on a website that the user is currently authenticated on (e.g. a ‘Submit’ button that will perform a particular action). To perform the attack, a malicious website will load a page from the website inside an iframe, using CSS to hide all except the targeted region of the page. The targeted region may either be displayed so that it appears to be part of the attacker’s site, a technique known as user interface (or UI) redressing; alternatively it may be made fully transparent and layered on top of another element on the site. JavaScript may also be used to position the iframe under the mouse cursor, such that the user will click on the target no matter where they click on the malicious page.

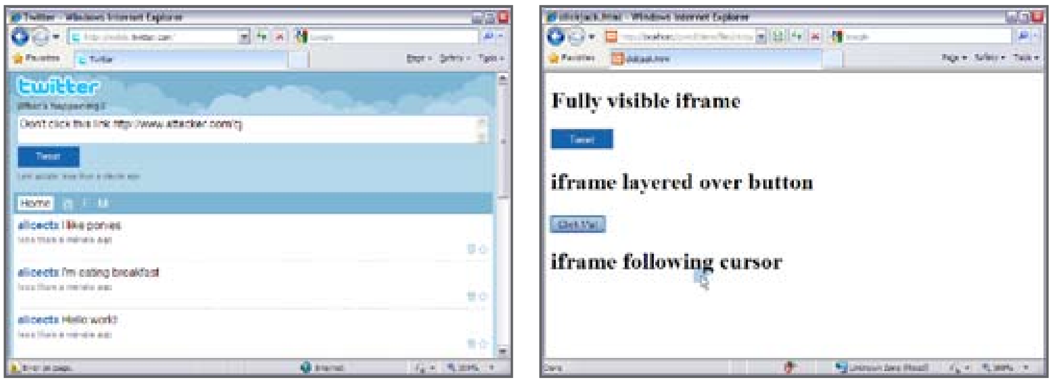


Figure 1
A normal web page and three ways in which it can be used for a clickjacking attack

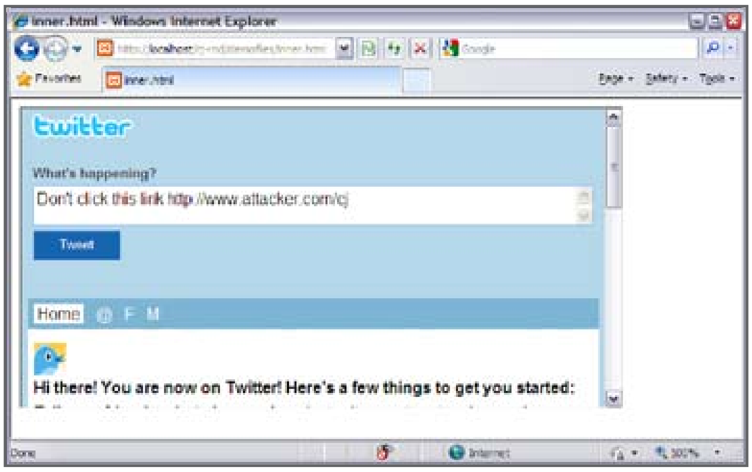
Various changes to web browser behaviour have been suggested to automatically prevent clickjacking attacks[2]. However, these methods have not been implemented by any major browser vendor as they are deemed tricky to implement effectively without breaking some legitimate uses of iframes. Therefore, the responsibility of protecting users from clickjacking attacks has fallen to website authors who must use either JavaScript or the X-Frame-Options HTTP[3] header in order to prevent their sites being loaded in iframes. The NoScript Firefox add-on provides automatic clickjacking protection for individual users.

High profile, in the wild clickjacking attacks have so far been limited to ‘nuisance’ viral worms, targeting social sites such as Twitter[4] and Facebook[5]. It is not known whether more malicious attacks are taking place, although a recent automated survey of over a million websites suggests that clickjacking attacks are not currently widespread[6].

It has been suggested that clickjacking will not become a popular tool for attackers until vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site-Request Forgery (CSRF) become less widespread in websites or are mitigated by browsers[7]. If a website is vulnerable to either XSS or CSRF, then clickjacking (as described above) is not necessary.

Basic Clickjacking

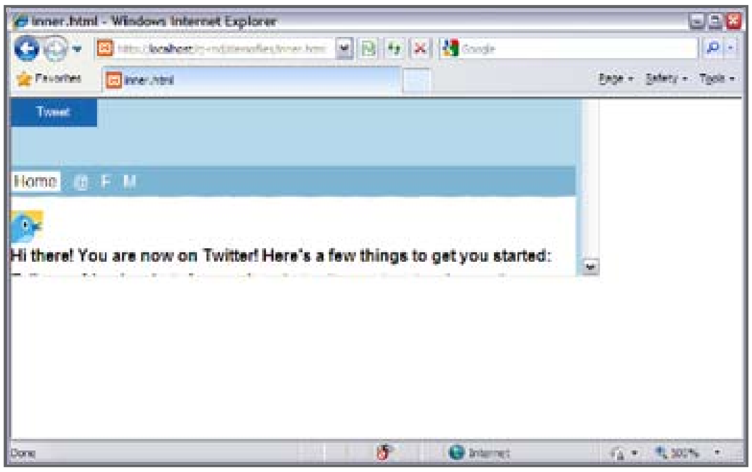
A typical clickjacking attack uses two nested iframes to crop and position an element from a target website. The inner iframe contains the target page and must be large enough to display it in its entirety, such that the element on which the user will click is visible without scrolling. The outer iframe is much smaller and acts as a window onto the page loaded in the inner iframe. For a UI redressing attack, the outer iframe should only be large enough to display the targeted element. For an attack using JavaScript and a moving invisible iframe, the outer iframe may be even smaller (e.g. 10-20 pixels square).



Don't click this link http://www.attacker.com/cj

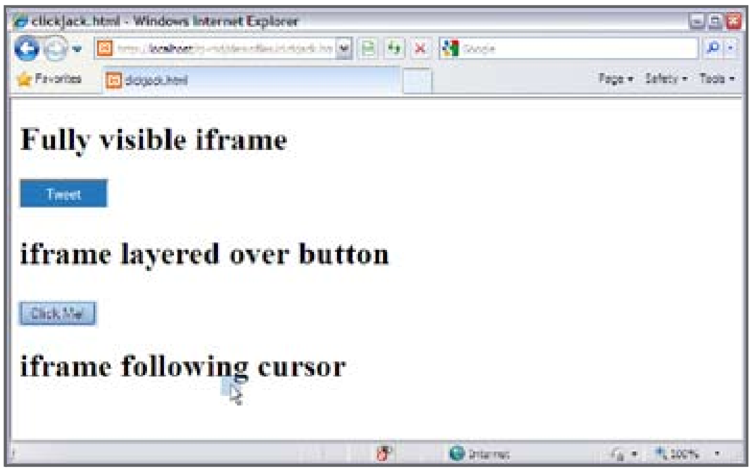
Hi there! You are now on Twitter! Here's a few things to get you started:

Step 1
A page (inner.html) with a large iframe is created, containing the target site



Hi there! You are now on Twitter! Here's a few things to get you started:

Step 2
The iframe is positioned so that the click target is at the top left of the page


clickjack.html - Windows Internet Explorer
Fully visible iframe
Tweet
iframe layered over button
Click Me!
iframe following cursor

Step 3
inner.html is loaded into a small iframe on clickjack.html. Here it can be positioned, layered on top of other elements, or made to follow the mouse cursor

The following HTML snippets show two iframes that will be referred to in examples throughout this paper:

```
<iframe id="inner" src="http://www.victim.com" width="1000" height="3000"
scrolling="no" frameborder="none"></iframe>
```

Figure 2
inner.html

```
<iframe id="outer" src="inner.html" width="20" height="20" scrolling="no"
frameborder="none"></iframe>
```

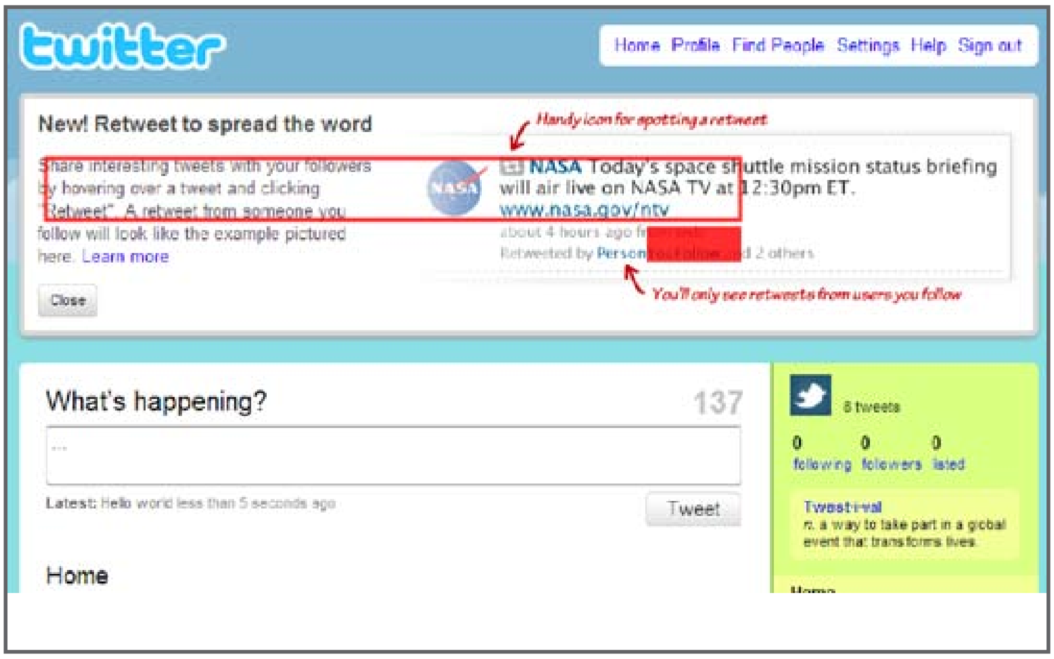
Figure 3
clickjack.html

Positioning Methods

In order to carry out a successful clickjacking attack, the targeted element (e.g. a button) must be carefully positioned. The inner iframe must be placed such that the element appears in the top left of inner.html. For example, if the element appears at the coordinates (400, 100) the inner iframe could be positioned with CSS as follows:

```
#inner { position: absolute; left: -400px; top: -100px }
```

However, on pages that contain dynamic content, the position of a target element may vary: content at the top of a page may push a click target further down a page. If the inner iframe is positioned based on fixed coordinates and the target page changes, then the user will click on a different part of the page, missing the target.


twitter
Home Profile Find People Settings Help Sign out
Handy icon for spotting a retweet
You'll only see retweets from users you follow
Close
137
Tweet
8 tweets
0 following
0 followers
0 listed
Home

New! Retweet to spread the word

Share interesting tweets with your followers by hovering over a tweet and clicking "Retweet". A retweet from someone you follow will look like the example pictured here. Learn more

NASA Today's space shuttle mission status briefing will air live on NASA TV at 12:30pm ET. www.nasa.gov/ntv

about 4 hours ago fr

Retweeted by Person d 2 others

What's happening?

Latest: Hello world less than 5 seconds ago

Tweest-val
n. a way to take part in a global event that transforms lives

Figure 4
An announcement box pushes a button further down a page. The red highlights show where the text box and button would have been.

Using pixel coordinates to position a target can also be inaccurate due to other factors, such as rendering differences between browsers and differing fonts between platforms.

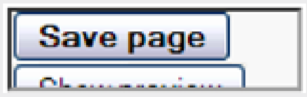
A solution to this problem is to use URL fragment identifiers[8] to position anchor elements in the inner iframe. Anchors and URL fragments are commonly used together to link to a particular section of text within an HTML document. When a URL containing a fragment identifier is loaded, a browser will scroll the page so that the anchor is at the top of the viewport[9]. An anchor can be created in two ways, either by adding a 'name' attribute to an 'a' tag, or by adding an 'id' attribute to any element:

```
<a name="foo"> - http://www.example.org/page.html#foo
<div id="foo"> - http://www.example.org/page.html#foo
```

Figure 5
Two types of page anchor, both can be referenced with a URL fragment identifier

Although the <a name> method is often used only for static documents, websites use ID attributes for various purposes, including CSS styling and JavaScript integration. HTML forms will often have an ID attribute on every element, including text fields and buttons.

For example, the 'Save' button on Wikipedia's 'Edit article' page has an ID of 'wpSave'. By loading the edit page into the inner iframe, and adding #wpSave on the end of the URL, the browser will scroll the outer iframe so that the button is visible.



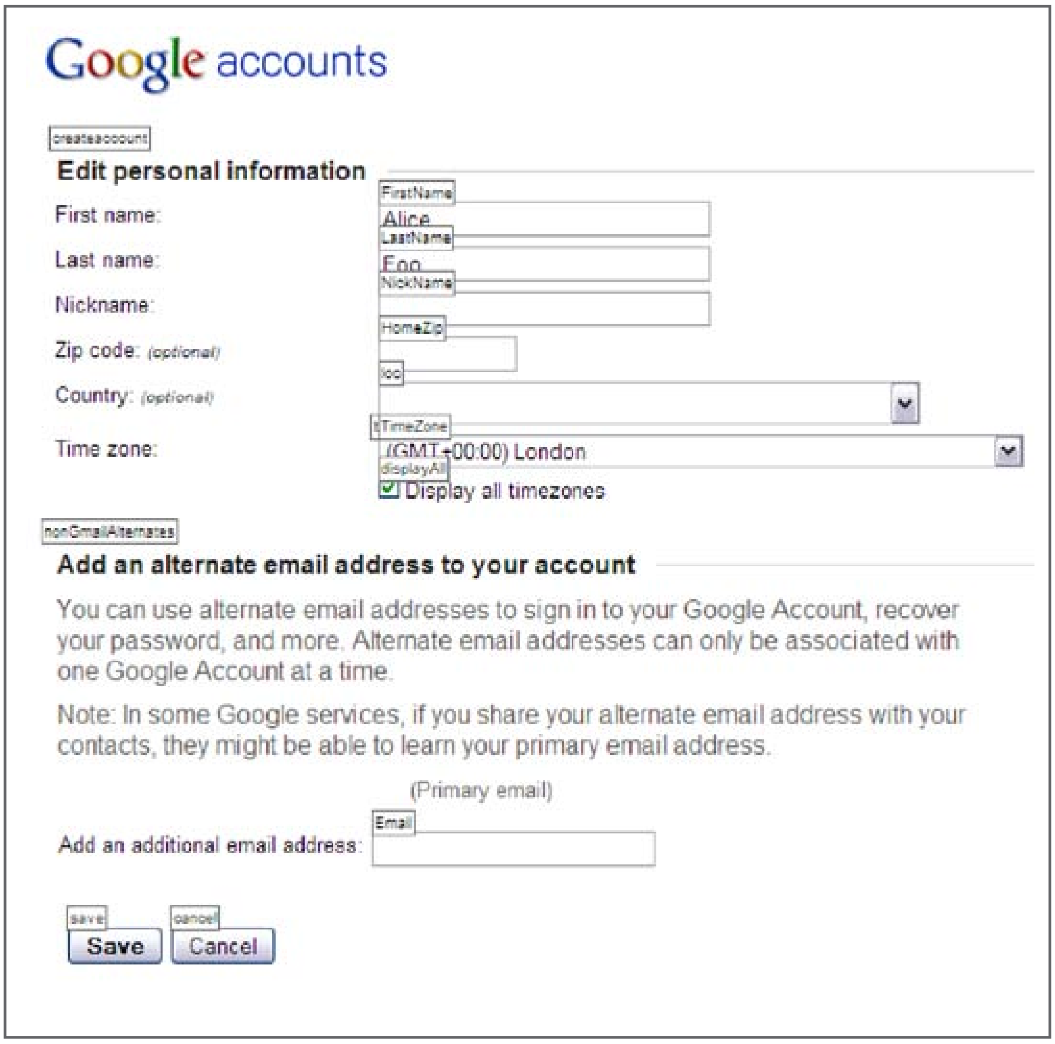
```
<iframe src="http://en.wikipedia.org/w/index.php?title=Clickjacking&action=ed
it#wpSave" width="130" height="34" scrolling="no"></iframe>
```

Figure 6
A click target positioned using a fragment identifier

Although scrollbars are disabled on both the inner and outer iframes, browsers will still 'scroll' the content of the iframes both horizontally and vertically, in order to make an anchor element visible.

Fragment positioning and pixel positioning may be combined if anchors are not present on the exact elements that are to be clicked. For example, a button may be at the bottom of a variable length page, 200 pixels above a footer that has an ID attribute. The page could be loaded with the footer fragment ID. The inner iframe can then be moved up by 200 pixels to make the button visible.

An alternative to using CSS for pixel positioning is to use the scrollTo and scrollBy methods to scroll the outer iframe. The scrollBy method can be used to perform relative positioning after fragment positioning has been used.



Google accounts

createaccount

Edit personal information

First name:	FirstName Alice
Last name:	LastName Foo
Nickname:	NickName
Zip code: *(optional)*	HomeZip
Country: *(optional)*	loc
Time zone:	tTimeZone (GMT+00:00) London
	displayAll ☑ Display all timezones

nonGmailAlternates

Add an alternate email address to your account

You can use alternate email addresses to sign in to your Google Account, recover your password, and more. Alternate email addresses can only be associated with one Google Account at a time.

Note: In some Google services, if you share your alternate email address with your contacts, they might be able to learn your primary email address.

(Primary email)

Add an additional email address: Email

Figure 7
Pages will often have many elements with IDs

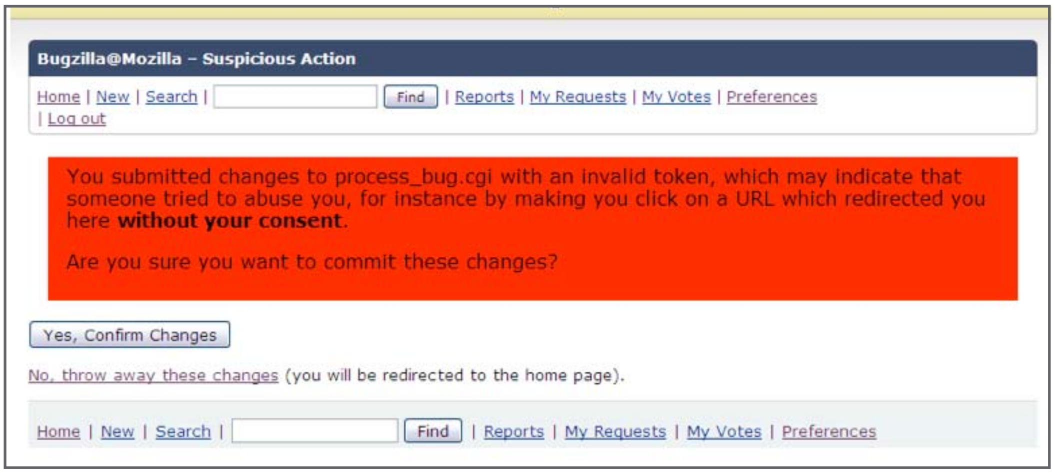
Clickjacking and Cross-Site Request Forgery

Clickjacking allows an attacker to bypass CSRF protections put in place by a website. The user is tricked into submitting a form directly from the website itself, so there is no need for the attacker to know hidden or secret values in the form, such as CSRF tokens.

However, clickjacking as it has currently been used is more limited than CSRF. Only clicks can be directed into a form, so while checkboxes and submit buttons can be clicked, an attacker cannot manipulate text fields. This is not a problem in some situations, since CSRF can often be used to prime a form with data, requiring only a single click to submit it. For example, the Twitter 'Don't Click' clickjacking worm took advantage of a feature that allows the status field to be prefilled by passing a URL parameter:

```
http://www.twitter.com/?status=foo
```

Many applications implement CSRF protection in a way that makes clickjacking straightforward. Often the CSRF token is checked as part of the form validation routine. If any field is invalid, (including the CSRF token), the form will be redisplayed with all the data provided, along with a valid CSRF token.



Bugzilla@Mozilla – Suspicious Action

You submitted changes to process_bug.cgi with an invalid token, which may indicate that someone tried to abuse you, for instance by making you click on a URL which redirected you here **without your consent**.

Are you sure you want to commit these changes?

No, throw away these changes (you will be redirected to the home page).

Home | New | Search | Find | Reports | My Requests | My Votes | Preferences | Log out
Yes, Confirm Changes
Home | New | Search | Find | Reports | My Requests | My Votes | Preferences

Figure 8
The Bugzilla issue tracker prevents CSRF but makes clickjacking easy

Many web applications do implement CSRF protection in a way that also prevents traditional clickjacking. In order to find a way to manipulate text fields, we must look at other ways in which a user can interact with a website.

Text Field Injection

All major desktop browsers allow drag-and-drop to be used as a way to move data around. Drag-and-drop is often used as an alternative to copy and paste, as it can be performed using a simple mouse gesture rather than menus or keyboard shortcuts. For example, text on a page can be selected and then dragged into a text field.

Most browsers support the drag-and-drop API (application programming interface), which was standardised as part of HTML5[10]. The API allows JavaScript on a web page to set data at the beginning of a drag operation, and allows dropped data to be read.

Drag-and-drop is not restricted by the 'same-origin policy'[11] that prevents a website from accessing data belonging to another domain. Data can be dragged freely from a website on one domain to another website. Browsers allow this because drag-and-drop operations must always be initiated by a user gesture, and cannot be started by JavaScript.

Despite these security precautions, drag-and-drop can be combined with clickjacking techniques to create a powerful new attack that allows arbitrary text to be entered into forms on another domain. The steps to carry out the attack are as follows:

1. A malicious website persuades a user to start dragging an item on the web page.

2. When the drag is started, the drag-and-drop API is used to set the drag data to the required text.

3. Once the drag has started, an invisible iframe is placed underneath the mouse cursor. The iframe contains a form on another website, positioned such that the mouse cursor is over a text field.

4. When the user drops the item, the attacker controlled text is entered into the form field.

These steps would be repeated for each text field as necessary, followed by a final click to submit the form. For more technical information on the events that are used to perform these steps, see Appendix 1 – Useful JavaScript Events.

A drag gesture is more complex to perform than a simple click, so a little more effort may be required to persuade a user to carry out the above steps. However, the attack could be presented under many different guises, including moving a slider or scrollbar, dragging products to a shopping cart or even moving pieces in a puzzle game. Since the attacker controls the position of both the drag source and drag target, the direction and distance of the drag does not matter. Provided the user drags the mouse at least a few pixels and releases the mouse button over the malicious page, the attack will succeed.

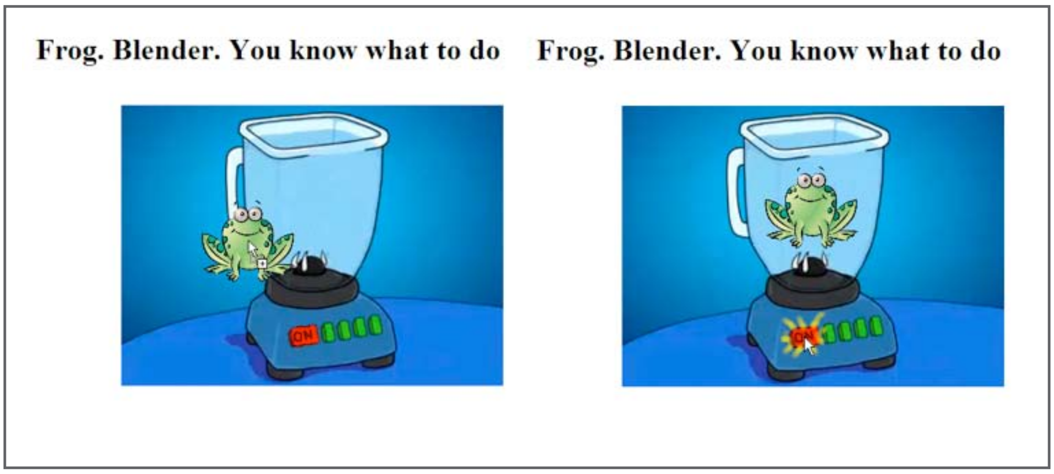

Frog. Blender. You know what to do
Frog. Blender. You know what to do
ON
ON

Figure 9
A drag followed by a click allows text to be injected into a form which is then submitted

This technique can be used in many situations where CSRF protection prevents the use of traditional clickjacking. It can also be used as a 'stepping-stone' for other types of attack that would have previously been difficult or impossible to carry out. For example, a site may be vulnerable to DOM (Document Object Model) based XSS'[12] through text entered into a search field. If the text can only be entered 'manually' then drag-and-drop could be used to deliver the XSS payload.

Content Extraction

A page cannot read the content of an iframe if it was loaded from a different domain, due to the same-origin policy. However, the drag-and-drop technique described above can be reversed in order to steal content and data from a framed website.

In order to extract content from a web page, we must first identify which items can be dragged. By default, all links and images on a page are draggable. When an image or link is dropped onto a page, it will be converted into a URL. While URLs are usually static, they will sometimes contain 'secret' data such as a document ID, a security token or data that could identify a user.

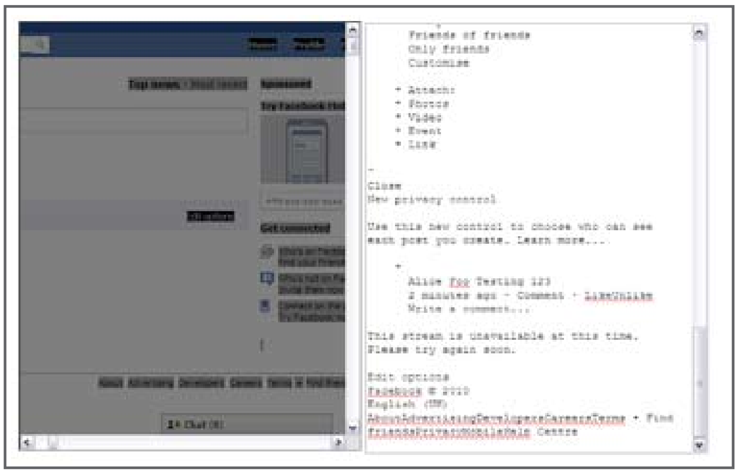
While URLs can be interesting, the textual content of a web page will often be much more valuable. It is possible to make arbitrary regions of any web page draggable, by creating a text selection. Selections are created by performing the same mouse gesture that is used for drag-and-drop operations - the mouse button is held down in one area of the page, moved until the desired content has been selected, and then released.

By using clickjacking techniques, an attacker can use an arbitrary user-initiated drag gesture to select a particular region of a framed web page. A second drag would then be used to extract the selection from the framed page and drop it onto the attacker's page.

The steps to carry this out are as follows:

Script Actions	User Actions
1. An invisible iframe is made to follow the user's mouse cursor.	
2. The document is positioned inside the iframe so that the mouse is over the area where the selection is to begin.	
	3. Begins a drag gesture, by holding down the mouse button.
4. The inner document is repositioned so that the mouse is over the area where the selection is to finish.	
	5. Moves the mouse (by at least one pixel), creating a selection between the two points.
	6. Releases the mouse button and moves the mouse outside of the iframe.
7. The document is positioned so that the mouse is over the selected area.	
	8. Performs a drag gesture, dragging the selection from inside the iframe and dropping it on the attacker's document.
9. The script may examine the content of the dropped data by calling the getData method.	

From the user's point of view, the above steps require only two drag operations. The start and end positions of the drags are not important, as the attacker can control both the position of the iframe and the document inside it. If an attacker can engage a user in a task that requires many drag operations (e.g. a sliding block puzzle game), the content of many web pages could be extracted.



```
        Friends of friends
        Only friends
        Customise

      • Attach:
      • Photos
      • Video
      • Event
      • Link

-
Close
New privacy control

Use this new control to choose who can see
each post you create. Learn more...

      •
        Alice Foo Testing 123
        2 minutes ago · Comment · LikeUnlike
        Write a comment...

This stream is unavailable at this time.
Please try again soon.

Edit options
Facebook © 2010
English (UK)
AboutAdvertisingDevelopersCareersTerms • Find
friendsPrivacyMobileHelp Centre
```

Figure 10
Content dragged from an iframe is extracted as plain text

Traditional clickjacking attacks must be typically aimed at web applications to which the attacker has access, so that the position of the required clicks can be determined and tested. The content extraction technique could be used in situations where the attacker knows the address of a web page but does not know its content, for example, a leaked intranet URL or an authenticated document. By using the steps above a selection of any size may be made, so it makes sense for an attacker to select the entire page. For an unknown document, an attacker can make a reasonable guess at the position to start dragging a selection (e.g. the top left of a page). The end point of the selection does not matter; if the user finishes dragging the selection well below the bottom of the page, the entire document will usually be selected.

This technique can also be used against documents rendered using browser plug-ins, for example, PDF documents.

HTML Source Extraction

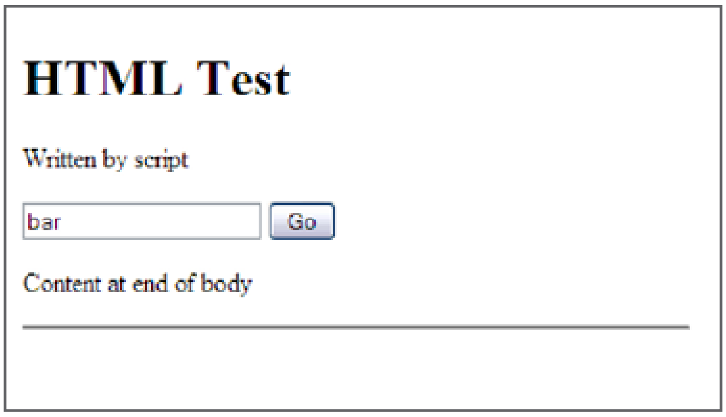
The content extraction technique as described will extract the visible content of a web page in plain text. While valuable and sensitive information may be gained using this method, the HTML source code of a web page will often contain further information of interest to an attacker.

All major browsers implement rich text editing capabilities which are used for applications such as webmail and document editing. An editable area may be made by setting the *designMode* property of an HTML document or by setting the *contentEditable* attribute of any HTML element[13]. Once enabled, a user may edit any HTML in these areas.

When a selection from an HTML document is dragged or pasted into an editable HTML area, the browser will first serialise the DOM of the selected content in the source document, and then recreate the elements inside the editable area. A script may access the source code of the editable area by reading its *innerHTML* property.

The steps to extract the HTML source for a page are identical to those in the previous section, except that in the final step, the script should position an invisible editable area underneath the cursor as the user drags the selection out of the iframe.

To view examples of the data can be extracted using this method, see the figures below. A simple HTML document was created, containing simple elements, inline scripts and a form with both hidden and visible fields. The entire page was selected using the mouse, then dragged to an editable HTML area. This was repeated in different browsers, and the resulting HTML source was extracted.


HTML Test

Written by script

Content at end of body

Figure 11
The rendered HTML document

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Drag Test</title>
<script>var a = 'inside head';</script>
</head>
<body>
<script>var b = 'inside body, before visible content';</script>
<h1>HTML Test</h1>
<script>document.write('<p>Written by script</p>');</script>
<form action="xyz">
<input type="hidden" value="secret">
<input type="text" name="foo" value="bar">
<input type="submit" value="Go">
</form>
<script>var c = 'inside body';</script>
<p id="an-id" class="a-class" nonstandard="attribute">Content at end of body</p>
<hr>
<script>var d = 'at end of body'</script>
</body>
</html>
```

Figure 12
The original HTML source

```
<h1>HTML Test</h1> <script>document.write('<p>Written by script</p>');</
script><p>Written by script</p> <form action="xyz"> <input value="secret"
type="hidden"> <input name="foo" value="bar" type="text"> <input value="Go"
type="submit"> </form> <script>var c = 'inside body';</script> <p id="an-id" class="a-
class" nonstandard="attribute">Content at end of body</p> <hr>
```

Figure 13
Copied source, Firefox 3.6.2

```
<H1>HTML Test</H1>
<SCRIPT>document.write('<p>Written by script</p>');</SCRIPT>

<P>Written by script</P>
<FORM action=xyz><INPUT value=secret type=hidden> <INPUT value=bar name=foo>
<INPUT value=Go type=submit> </FORM>
<SCRIPT>var c = 'inside body';</SCRIPT>

<P id=an-id class=a-class nonstandard="attribute">Content at end of body</P>
```

Figure 14
Copied source, Internet Explorer 8.0.6001.18702

```
<h1>HTML Test</h1><p>Written by script</p><form action="http://example.com/
xyz"><input type="text" name="foo" value="bar">&nbsp;<input type="submit"
value="Go"></form><p id="an-id" class="a-class" nonstandard="attribute">Content
at end of body</p><div><br></div>
```

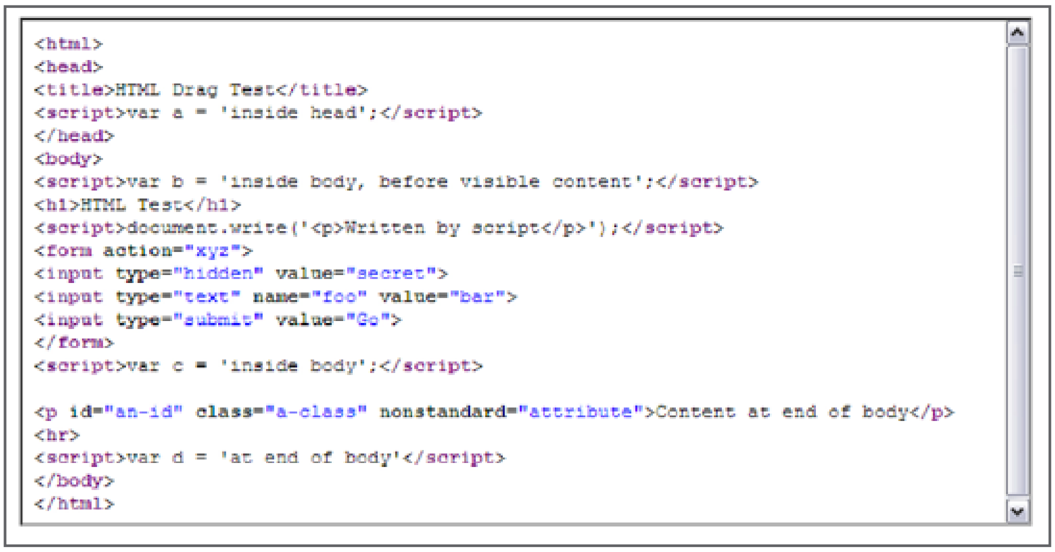
Figure 15
Copied source, Chrome 5.0.342.8 beta

In Internet Explorer and Firefox, all content between the first and last visible elements was copied, including hidden form fields and the content of script tags. Script tags at the very beginning and end of the document and content in the head tag were not copied. In Chrome (and other WebKit based browsers) only visible content was copied, although attributes on those elements including IDs, classes and URLs were copied. In all browsers, elements that have been created dynamically are also copied.

The source of an HTML document may contain several items of interest to an attacker. Many web applications implement CSRF protection by including a hidden field in each form. The value of the hidden field is a random token that cannot be guessed by an attacker. However, if this token is stolen, the CSRF protection is effectively broken. If the CSRF token is reusable, an attacker may perform a fully automated CSRF attack consisting of many page requests. For example, if a reusable CSRF token for a webmail application were stolen an attacker could send emails from the user's account or forward emails in the user's inbox to the attacker's email address.

In addition to CSRF tokens, a document may contain sensitive URLs and various other sensitive data within HTML attributes.

The Firefox browser allows a second kind of HTML source theft. The view-source pseudo-protocol can be used to load the HTML source text of a document instead of rendering it visually. Any document may be loaded into an iframe using the view-source protocol. For example, instead of loading *http://www.example.com* inside the iframe, the URL *view-source:http://www.example.com* would be loaded instead. Although the same-origin policy is enforced (preventing script access to the content of the frame) an attacker may obtain the source using the same content extraction method as described above. This method has the benefit that the entire HTML source may be obtained, including the head element. Additionally, the document is a static text file, so no JavaScript is run. This defeats any script-based anti-framing code that is included on the page.



```
<html>
<head>
<title>HTML Drag Test</title>
<script>var a = 'inside head';</script>
</head>
<body>
<script>var b = 'inside body, before visible content';</script>
<h1>HTML Test</h1>
<script>document.write('<p>Written by script</p>');</script>
<form action="xyz">
<input type="hidden" value="secret">
<input type="text" name="foo" value="bar">
<input type="submit" value="Go">
</form>
<script>var c = 'inside body';</script>

<p id="an-id" class="a-class" nonstandard="attribute">Content at end of body</p>
<hr>
<script>var d = 'at end of body'</script>
</body>
</html>
```

Figure 16
An iframe containing a document loaded using view-source, inside Firefox

Forced Drag-and-Drop with Java Applets

When combined with clickjacking techniques, drag-and-drop can allow for interesting new techniques, as has been seen with the text field injection and content extraction methods. However, drag-and-drop gestures require the user to perform actions not commonly used in normal browsing. The minimum required interaction to perform a drag-and-drop operation is a mouse down event, followed by a mouse move event, followed by a mouse up.

Java provides a much richer drag-and-drop API than is implemented by web browsers[14], and allows the standard behaviour to be overridden. The API contains a MouseDragGestureRecogniser class that observes mouse events and triggers a drag event when the correct mouse gesture is performed. An unprivileged Java applet may override this class, changing the type of interaction that is required to begin a drag operation. For example, a drag may be initiated by only a mouse down event. When the mouse button is released, the drop operation is completed, causing attacker controlled text to be inserted into any text field underneath the user's cursor. This allows a drag-and-drop event to be forced as a result of a normal click.

If Java is available in a browser, the text field injection technique can be simplified to require only mouse clicks, instead of drag gestures.

The Java drag-and-drop API may be further abused, by initiating a drag-and-drop operation with no user interaction whatsoever. This is possible even when the mouse cursor is not over the applet. If the user's mouse button is not held down when the drag operation begins, the drop will complete immediately. A script on an attacker's web page can coordinate with a Java applet to fill several form fields on another domain, without user interaction. The page script would use an invisible iframe to position each text field in turn underneath the mouse cursor, using the applet to initiate a drop for each field. Only a final click would be required to submit the form.

The Java API also allows any mouse cursor to be displayed during the drag-and-drop operation. By using a default mouse cursor instead of the standard 'drag' mouse cursor, the user is given no visual indication what is going on.

This technique allows many form fields to be filled without user interaction, giving rise to many new types of attack. For example, an attacker could add a new default shipping address to a user's account on a shopping website, filling in several form fields automatically and requiring just a single click to save the address.

The forced drag-and-drop method has been tested using version 6 of the Sun Java Runtime on Windows and the version 5 of the MacOS X Java Runtime. On Linux, a real drag gesture is still required to complete a drop.

Anchor Element Position Detection

Browser based attacks such as clickjacking, CSRF and XSS usually require a victim to be authenticated to a particular web application in order to succeed. An attacker will often wish to know whether a user is authenticated against a particular application before carrying out an attack. For techniques such as clickjacking, where successful exploitation requires user interaction, an attacker will want to maximise the effectiveness of every mouse click or movement. For example, a number of webmail services may be vulnerable to clickjacking, but a user may be authenticated to only one. The chances of a successful attack will be increased by targeting only the application to which the user is authenticated.

Many browser based login detection techniques have been described, including timing based attacks[15], and loading authenticated resources such as style sheets[16] or images[17]. However, these methods are highly dependent on the implementation of each application. The technique described below is more general and can be easily adapted for almost any web application.

The fragment identifier positioning method described earlier can be used to leak information about a page loaded in an iframe. Two iframes are used to position a document for a clickjacking attack – an iframe containing the target document is contained within a smaller iframe that acts as a window onto a particular element. When a URL containing a fragment identifier is loaded in the inner iframe, the browser will scroll the corresponding anchor into view. However, the inner iframe is large (big enough to fit the entire content of the target document) and cannot be scrolled. Therefore it is the smaller, outer iframe that is scrolled. The content of the outer iframe belongs to the attacker's domain, allowing its scroll position to be read by JavaScript. If the loaded URL has a fragment identifier that does not correspond to an anchor on the page, then the scroll position of the iframe will not change.

This behaviour is consistent across all major browsers, including Internet Explorer, Firefox, Chrome and Safari.

```
var outer = document.getElementById('outer');
var inner = outer.contentWindow.document.getElementById('inner');
inner.src = ,http://www.victim.com/myprofile#username';
var x, y;

if ('scrollX' in inner.contentWindow) {
   x = inner.contentWindow.scrollX;
   y = inner.contentWindow.scrollY;
} else {
   x = inner.contentWindow.document.documentElement.scrollLeft;
   x = inner.contentWindow.document.documentElement.scrollTop;
}
```

Figure 17 - Determining the position of an anchor element

Figure 17
Determining the position of an anchor element

This technique can be used to perform login detection by checking for the presence of a particular element that exists only on authenticated (or unauthenticated) pages. For example, web applications will often redirect to a login screen if a URL that requires authentication is loaded.

The Google Account login page has the IDs 'Email' and 'Passwd'. An attacker could attempt to load an authenticated page such as *https://www.google.com/accounts/ManageAccount*, and then check for the presence of one or both of these IDs to determine if the login screen was loaded in the iframe.

While this technique is well suited for aiding a clickjacking attack due to the use of iframes, it can be used in many other situations that are beyond the scope of this paper. One such example is a document repository that allows complex search queries to be performed. If the search results page has a footer element with an ID, then the length of the page can be determined. By performing search queries for different terms (e.g. 'starts with a', 'starts with b', 'starts with ab'), an automated script could determine how many documents match a particular search. A binary search would eventually reveal the titles of documents within the system.

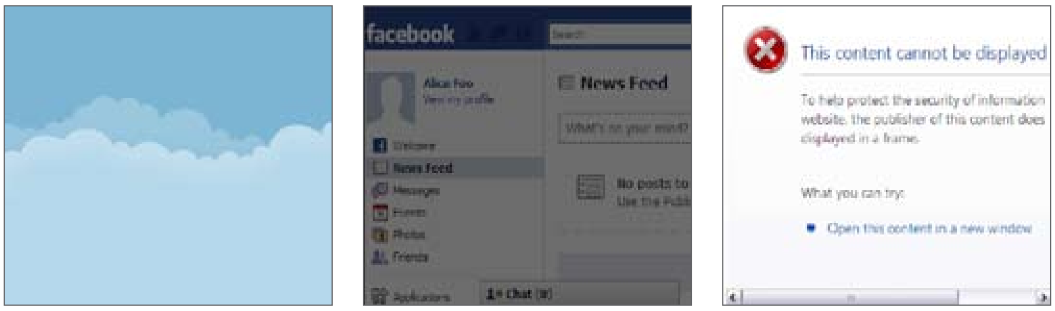
Clickjacking Defences

While a number of new techniques have been described in this paper, they can mostly be defeated by the same methods that protect against traditional clickjacking.

Frame-busting[18] was the first technique that was recommended to counter clickjacking attacks. A page using this method will detect that is has been framed by another web site, and attempt to load itself in place of the site that is framing it (thus 'busting out' of the frame). However, a malicious site may try to use the *onunload* and *onbeforeunload* page events to prevent a framed site from navigating to a different URL[19].

An alternative to frame-busting is for a page to simply hide or obscure its content if it detects that it is being framed. Both Twitter and Facebook now use this approach. When framed, Twitter will hide its content and attempt to frame-bust. Facebook takes a slightly different approach by placing a semi-transparent overlay over its page, and will frame-bust when the page is clicked.

No JavaScript based method of clickjacking protection should be deemed 100 percent effective[20], and as a result browser vendors are now implementing declarative methods such as X-Frame-Options[3], first introduced by Microsoft in Internet Explorer 8. Web browsers that support this security feature will prevent a web page being displayed in an iframe if the X-Frame-Options header is set by the page. In order to protect older browsers that do not support this feature, it is advisable for sites to use X-Frame-Options in addition to JavaScript-based methods.



This content cannot be displayed

To help protect the security of information website, the publisher of this content does displayed in a frame.

What you can try:

Open this content in a new window

Figure 18
Framed Twitter and Facebook pages, and a page using X-Frame-Options in Internet Explorer

Conclusion

Traditional clickjacking is a powerful technique, but there are many situations in which it cannot be used. The methods described in this paper – text field injection, content extraction, HTML source extraction, forced drag-and-drop and anchor leakage – build upon clickjacking and potentially pose a threat to many more types of web application.

About Context

Context Information Security is an independent security consultancy specialising in both technical security and information assurance services

The company was founded in 1998. Its client base has grown steadily over the years, thanks in large part to personal recommendations from existing clients who value us as business partners. We believe our success is based on the value our clients place on our product-agnostic, holistic approach; the way we work closely with them to develop a tailored service; and to the independence, integrity and technical skills of our consultants.

The company's client base now includes some of the most prestigious blue chip companies in the world, as well as government organisations.

The best security experts need to bring a broad portfolio of skills to the job, so Context has always sought to recruit staff with extensive business experience as well as technical expertise. Our aim is to provide effective and practical solutions, advice and support: when we report back to clients we always communicate our findings and recommendations in plain terms at a business level as well as in the form of an in-depth technical report.

context
INFORMATION SECURITY LTD

Acknowledgements

Thanks to Robert Hansen (RSnake) and Jeremiah Grossman for their original clickjacking research. Many thanks also go to my colleagues at Context who have supported this research.

References

1.
Robert Hansen and Jeremiah Grossman. (2008, Sep.)
Explanation of Clickjacking. [Online].
http://www.sectheory.com/clickjacking.htm

2.
Michal Zalewski. (2008, September) Dealing with UI redress vulnerabilities inherent
to the current web, WHATWG Mailing List. [Online].
http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2008-September/016284.html

3.
Eric Lawrence. (2009, January) IE8 Security Part VII: ClickJacking Defenses. [Online].
http://blogs.msdn.com/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx

4.
The Register. (2009, February) Twitter attack exposes awesome power of clickjacking [Online].
http://www.theregister.co.uk/2009/02/13/twitter_clickjack_attack/

5.
Joey Tyson. (2009, November) Facebook Worm Uses Clickjacking in the Wild. [Online].
http://theharmonyguy.com/2009/11/23/facebook-worm-uses-clickjacking-in-the-wild/

6.
Marco Balduzzi, Manuel Egele, Davide Balzarotti, Engin Kirda, and Christopher Kruegel,
"A Solution for the Automated Detection of Clickjacking Attacks," in ASIACCS'10, Beijing.

7.
Jeremiah Grossman. (2009, June) Clickjacking 2017. [Online].
http://jeremiahgrossman.blogspot.com/2009/06/clickjacking-2017.html

8.
W3C. HTML 4.01 Specification, Introduction to links and anchors. [Online].
http://www.w3.org/TR/REC-html40/struct/links.html#anchors

9.
W3C. HTML5 Specification, Session history and navigation. [Online].
http://www.w3.org/TR/html5/history.html#scroll-to-fragid

10.
W3C. HTML5 Specification, Drag and Drop. [Online].
http://www.w3.org/TR/html5/editing.html#dnd

11.
Michal Zalewski. Browser Security Handbook. [Online].
http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy

12.
Amit Klein. (2005, July) DOM Based Cross Site Scripting. [Online].
http://www.webappsec.org/projects/articles/071105.html

13.
W3C. HTML5 Specification, The contenteditable attribute. [Online].
http://www.w3.org/TR/html5/editing.html#contenteditable

14.
Oracle. Drag and Drop Subsystem for the JavaTM 2 Platform Standard Edition 5.0. [Online].
http://java.sun.com/javase/6/docs/technotes/guides/dragndrop/spec/dnd1.html

15.
Andrew Bortz, Dan Boneh, and Palash Nandy, "Exposing Private Information by Timing Web Applications," in WWW 2007, Banff, Alberta, Canada, pp. 621-628.

16. Chris Evans. (2008, August) Cross-domain leaks of site logins. [Online].
http://scarybeastsecurity.blogspot.com/2008/08/cross-domain-leaks-of-site-logins.html

17.
Jeremiah Grossman. (2008, March) Login Detection, whose problem is it? [Online].
http://jeremiahgrossman.blogspot.com/2008/03/login-detection-whose-problem-is-it.html

18.
Wikipedia. Framekiller. [Online].
http://en.wikipedia.org/wiki/Framekiller

19.
coderrr. (2009, February) Preventing Frame Busting and Click Jacking (UI Redressing). [Online].
http://coderrr.wordpress.com/2009/02/13/preventing-frame-busting-and-click-jacking-ui-redressing/

20.
Mozilla Bugzilla. IFRAME inside designMode disables JavaScript, breaking current clickjacking defenses. [Online].
https://bugzilla.mozilla.org/show_bug.cgi?id=519928

Appendix 1 – Useful JavaScript Events

The following events are useful for clickjacking, text field injection and content extraction.

focus – In Internet Explorer, the focus event will be fired on an iframe when the user clicks the mouse on it, or holds down the mouse button at the start of a drag operation.

blur – This is used to detect iframe focus in browsers other than Internet Explorer. The blur event will be fired on the document object when another document or window receives focus. The blur event should be used in conjunction with the *mouseover* and *mouseout* in order to determine whether the mouse is over the iframe when focus is lost (i.e. whether the user clicked in the iframe or elsewhere)

After an iframe has been focused by a user, the document that contains the iframe will not receive any more focus or blur events until the iframe loses focus. In order to detect multiple clicks in an iframe, a script can call *window.focus()* to regain focus.

dragStart – Used for text field injection. The *dataTransfer.setData* method can be used to set the text that will be entered into the text field

dragOver – Can be used for both text field injection and content extraction. The event contains the position of the mouse and can be used to update the position of a visual item being dragged, or to place an invisible drop target underneath the mouse cursor.

dragEnd – This is fired when a drag operation completes or is cancelled, even in another document. When used with the *mouseover* and *mouseout* events, a script can determine whether a drop occurred over an iframe or elsewhere. This is handy when performing text field injection.

drop – This is fired when the user drops an item (e.g. selection dragged from an iframe) on a document . The *dataTransfer.getData* method can be used to retrieve the plain text of the dropped content. Scripts that wish to access dropped data as HTML should wait for the drop event before accessing the *innerHTML* attribute of the *designMode* document or *contentEditable* element onto which it was dropped.

mousemove – This is fired every time the mouse is moved and can be used to keep an invisible iframe placed underneath the mouse cursor. Internet Explorer will also fire this event on a containing document when the user is dragging a selection inside an iframe, while the mouse is moving outside the bounds of the iframe. Other browsers do not send any mouse events to a parent document while a selection is being dragged inside an iframe.

onload – The *onload* event will fire on an iframe when the document inside it has loaded.