# Clickjacking

## What is clickjacking

Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online.

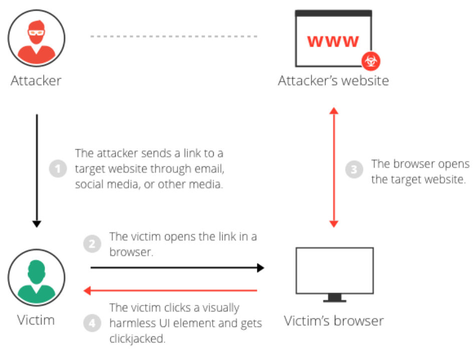
Typically, clickjacking is performed by displaying an invisible page or HTML element, inside an iframe, on top of the page the user sees. The user believes they are clicking the visible page but in fact they are clicking an invisible element in the additional page transposed on top of it.

The invisible page could be a malicious page, or a legitimate page the user did not intend to visit – for example, a page on the user's banking site that authorizes the transfer of money.There are several variations of the clickjacking attack, such as:

- **Likejacking** – a technique in which the Facebook "Like" button is manipulated, causing users to "like" a page they actually did not intend to like.
- **Cursorjacking** – a UI redressing technique that changes the cursor for the position the user perceives to another position. Cursorjacking relies on vulnerabilities in Flash and the Firefox browser, which have now been fixed.

## Clickjacking attack example

1. The attacker creates an attractive page which promises to give the user a free trip to Tahiti.
2. In the background the attacker checks if the user is logged into his banking site and if so, loads the screen that enables transfer of funds, using query parameters to insert the attacker's bank details into the form.
3. The bank transfer page is displayed in an invisible iframe above the free gift page, with the "Confirm Transfer" button exactly aligned over the "Receive Gift" button visible to the user.
4. The user visits the page and clicks the "Book My Free Trip" button.
5. In reality the user is clicking on the invisible iframe, and has clicked the "Confirm Transfer" button. Funds are transferred to the attacker.
6. The user is redirected to a page with information about the free gift (not knowing what happened in the background).


Attacker
www
Attacker's website
1 The attacker sends a link to a target website through email, social media, or other media.
3 The browser opens the target website.
2 The victim opens the link in a browser.
Victim
4 The victim clicks a visually harmless UI element and gets clickjacked.
Victim's browser

This example illustrates that, in a clickjacking attack, the malicious action (on the bank website, in this case) cannot be traced back to the attacker because the user performed it while being legitimately signed into their own account.

## Clickjacking mitigation

There are two general ways to defend against clickjacking:

- **Client-side methods** – the most common is called Frame Busting. Client-side methods can be effective in some cases, but are considered not to be a best practice, because they can be easily bypassed.
- **Server-side methods** – the most common is X-Frame-Options. Server-side methods are recommended by security experts as an effective way to defend against clickjacking.

## Mitigating clickjacking with X-Frame-Options response header

The X-Frame-Options response header is passed as part of the HTTP response of a web page, indicating whether or not a browser should be allowed to render a page inside a <FRAME> or <IFRAME> tag.

There are three values allowed for the X-Frame-Options header:

- **DENY** – does not allow any domain to display this page within a frame
- **SAMEORIGIN** – allows the current page to be displayed in a frame on another page, but only within the current domain
- **ALLOW-FROM URI** – allows the current page to be displayed in a frame, but only in a specific URI – for example *www.example.com/frame-page*

## Using the SAMEORIGIN option to defend against clickjacking

X-Frame-Options allows content publishers to prevent their own content from being used in an invisible frame by attackers.

The DENY option is the most secure, preventing any use of the current page in a frame. More commonly, SAMEORIGIN is used, as it does enable the use of frames, but limits them to the current domain.

### Limitations of X-Frame-Options

- To enable the SAMEORIGIN option across a website, the X-Frame-Options header needs to be returned as part of the HTTP response for each individual page (cannot be applied cross-site).
- X-Frame-Options does not support a whitelist of allowed domains, so it doesn't work with multi-domain sites that need to display framed content between them.
- Only one option can be used on a single page, so, for example, it is not possible for the same page to be displayed as a frame both on the current website and an external site.
- The ALLOW-FROM option is not supported by all browsers.
- X-Frame-Options is a deprecated option in most browsers.

### Clickjacking test – Is your site vulnerable?

A basic way to test if your site is vulnerable to clickjacking is to create an HTML page and attempt to include a sensitive page from your website in an iframe. It is important to execute the test code on another web server, because this is the typical behavior in a clickjacking attack.

Use code like the following, provided as part of the OWASP Testing Guide:

```
<html>
<head>
<title>Clickjack test page</title>
</head>
<body>
<p>Website is vulnerable to clickjacking!</p>
<iframe src="http://www.yoursite.com/sensitive-page" width="500" height="500">
</iframe>
</body>
</html>
```

View the HTML page in a browser and evaluate the page as follows:

- If the text "Website is vulnerable to clickjacking" appears and below it you see the content of your sensitive page, **the page is vulnerable to clickjacking**.
- If only the text "Website is vulnerable to clickjacking" appears, and you do not see the content of your sensitive page, the page is not vulnerable to the simplest form of clickjacking.

However, additional testing is needed to see which anti-clickjacking methods are used on the page, and whether they can be bypassed by attackers.

**How Imperva helps mitigate clickjacking attack**

To get to the point of clickjacking a site, the site will have to be compromised, something Imperva WAF prevents. You should also make sure your site resources are sending the proper X-Frame-Options HTTP headers, which would prevent some parts of your site from being framed in other pages or outside your domain.

# Clickjacking (UI redressing)

In this section we will explain what clickjacking is, describe common examples of clickjacking attacks and discuss how to protect against these attacks.

## What is clickjacking?

Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website. Consider the following example:

A web user accesses a decoy website (perhaps this is a link provided by an email) and clicks on a button to win a prize. Unknowingly, they have been deceived by an attacker into pressing an alternative hidden button and this results in the payment of an account on another site. This is an example of a clickjacking attack. The technique depends upon the incorporation of an invisible, actionable web page (or multiple pages) containing a button or hidden link, say, within an iframe. The iframe is overlaid on top of the user's anticipated decoy web page content. This attack differs from a CSRF attack in that the user is required to perform an action such as a button click whereas a CSRF attack depends upon forging an entire request without the user's knowledge or input.

Protection against CSRF attacks is often provided by the use of a CSRF token: a session-specific, single-use number or nonce. Clickjacking attacks are not mitigated by the CSRF token as a target session is established with content loaded from an authentic website and with all requests happening on-domain. CSRF tokens are placed into requests and passed to the server as part of a normally behaved session. The difference compared to a normal user session is that the process occurs within a hidden iframe.

*Labs*
If you're already familiar with the basic concepts behind clickjacking vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

View all clickjacking labs

## How to construct a basic clickjacking attack

Clickjacking attacks use CSS to create and manipulate layers. The attacker incorporates the target website as an iframe layer overlaid on the decoy website. An example using the style tag and parameters is as follows:

```
<head>
	<style>
		#target_website {
			position:relative;
			width:128px;
			height:128px;
			opacity:0.00001;
			z-index:2;
			}
		#decoy_website {
			position:absolute;
			width:300px;
			height:400px;
			z-index:1;
			}
	</style>
</head>
```

```
...
```

```
<body>
```

```
        <div id="decoy_website">
```

```
        ...decoy web content here...
```

```
        </div>
```

```
        <iframe id="target_website" src="https://vulnerable-website.com">
```

```
        </iframe>
```

```
</body>
```

The target website iframe is positioned within the browser so that there is a precise overlap of the target action with the decoy website using appropriate width and height position values. Absolute and relative position values are used to ensure that the target website accurately overlaps the decoy regardless of screen size, browser type and platform. The z-index determines the stacking order of the iframe and website layers. The opacity value is defined as 0.0 (or close to 0.0) so that the iframe content is transparent to the user. Browser clickjacking protection might apply threshold-based iframe transparency detection (for example, Chrome version 76 includes this behavior but Firefox does not). The attacker selects opacity values so that the desired effect is achieved without triggering protection behaviors.

### Clickbandit

Although you can manually create a clickjacking proof of concept as described above, this can be fairly tedious and time-consuming in practice. When you're testing for clickjacking in the wild, we recommend using Burp's Clickbandit tool instead. This lets you use your browser to perform the desired actions on the frameable page, then creates an HTML file containing a suitable clickjacking overlay. You can use this to generate an interactive proof of concept in a matter of seconds, without having to write a single line of HTML or CSS.

## Clickjacking with prefilled form input

Some websites that require form completion and submission permit prepopulation of form inputs using GET parameters prior to submission. Other websites might require text before form submission. As GET values form part of the URL then the target URL can be modified to incorporate values of the attacker's choosing and the transparent "submit" button is overlaid on the decoy site as in the basic clickjacking

## Frame busting scripts

Clickjacking attacks are possible whenever websites can be framed. Therefore, preventative techniques are based upon restricting the framing capability for websites. A common client-side protection enacted through the web browser is to use frame busting or frame breaking scripts. These can be implemented via proprietary browser JavaScript add-ons or extensions such as NoScript. Scripts are often crafted so that they perform some or all of the following behaviors:

- check and enforce that the current application window is the main or top window,
- make all frames visible,
- prevent clicking on invisible frames,
- intercept and flag potential clickjacking attacks to the user.

Frame busting techniques are often browser and platform specific and because of the flexibility of HTML they can usually be circumvented by attackers. As frame busters are JavaScript then the browser's security settings may prevent their operation or indeed the browser might not even support JavaScript. An effective attacker workaround against frame busters is to use the HTML5 iframe `sandbox` attribute. When this is set with the `allow-forms` or `allow-scripts` values and the `allow-top-navigation` value is omitted then the frame buster script can be neutralized as the iframe cannot check whether or not it is the top window:

```
<iframe id="victim_website" src="https://victim-website.com" sandbox="allow-
forms"></iframe>
```

Both the `allow-forms` and `allow-scripts` values permit the specified actions within the iframe but top-level navigation is disabled. This inhibits frame busting behaviors while allowing functionality within the targeted site.

## Combining clickjacking with a DOM XSS attack

So far, we have looked at clickjacking as a self-contained attack. Historically, clickjacking has been used to perform behaviors such as boosting "likes" on a Facebook page. However, the true potency of clickjacking is revealed when it is used as a carrier for another attack such as a DOM XSS attack. Implementation of this combined attack is relatively straightforward assuming that the attacker has first identified the XSS exploit. The XSS exploit is then combined with the iframe target URL so that the user clicks on the button or link and consequently executes the DOM XSS attack.

## Multistep clickjacking

Attacker manipulation of inputs to a target website may necessitate multiple actions. For example, an attacker might want to trick a user into buying something from a retail website so items need to be added to a shopping basket before the order is placed. These actions can be implemented by the attacker using multiple divisions or iframes. Such attacks require considerable precision and care from the attacker

## How to prevent clickjacking attacks

We have discussed a commonly encountered browser-side prevention mechanism, namely frame busting scripts. However, we have seen that it is often straightforward for an attacker to circumvent these protections. Consequently, server driven protocols have been devised that constrain browser iframe usage and mitigate against clickjacking.

Clickjacking is a browser-side behavior and its success or otherwise depends upon browser functionality and conformity to prevailing web standards and best practice. Server-side protection against clickjacking is provided by defining and communicating constraints over the use of components such as iframes. However, implementation of protection depends upon browser compliance and enforcement of these constraints. Two mechanisms for server-side clickjacking protection are X-Frame-Options and Content Security Policy.

### X-Frame-Options

X-Frame-Options was originally introduced as an unofficial response header in Internet Explorer 8 and it was rapidly adopted within other browsers. The header provides the website owner with control over the use of iframes or objects so that inclusion of a web page within a frame can be prohibited with the `deny` directive:

```
X-Frame-Options: deny
```

Alternatively, framing can be restricted to the same origin as the website using the `sameorigin` directive

```
X-Frame-Options: sameorigin
```

or to a named website using the `allow-from` directive:

```
X-Frame-Options: allow-from https://normal-website.com
```

X-Frame-Options is not implemented consistently across browsers (the `allow-from` directive is not supported in Chrome version 76 or Safari 12 for example). However, when properly applied in conjunction with Content Security Policy as part of a multi-layer defense strategy it can provide effective protection against clickjacking attacks.

## Content Security Policy (CSP)

Content Security Policy (CSP) is a detection and prevention mechanism that provides mitigation against attacks such as XSS and clickjacking. CSP is usually implemented in the web server as a return header of the form:

```
Content-Security-Policy: policy
```

where policy is a string of policy directives separated by semicolons. The CSP provides the client browser with information about permitted sources of web resources that the browser can apply to the detection and interception of malicious behaviors.

The recommended clickjacking protection is to incorporate the `frame-ancestors` directive in the application's Content Security Policy. The `frame-ancestors 'none'` directive is similar in behavior to the X-Frame-Options `deny` directive. The `frame-ancestors 'self'` directive is broadly equivalent to the X-Frame-Options `sameorigin` directive. The following CSP whitelists frames to the same domain only:

```
Content-Security-Policy: frame-ancestors 'self';
```

Alternatively, framing can be restricted to named sites:

```
Content-Security-Policy: frame-ancestors normal-website.com;
```

To be effective against clickjacking and XSS, CSPs need careful development, implementation and testing and should be used as part of a multi-layer defense strategy.