Clickjacking

⋮ 20/07/2022

Jul 19, 2022

9 min read
Click-Jacking Web-Notes

References

Clickjacking Defense - OWASP Cheat Sheet Series

Web Application Penetration Testing Notes

Clickjacking - Kontra

Protecting Your Users Against Clickjacking

Clickjacking

GitHub - alexbieber/Bug_Bounty_writeups: BUG BOUNTY WRITEUPS - OWASP TOP 10 🔴🔴🔴🔴✔

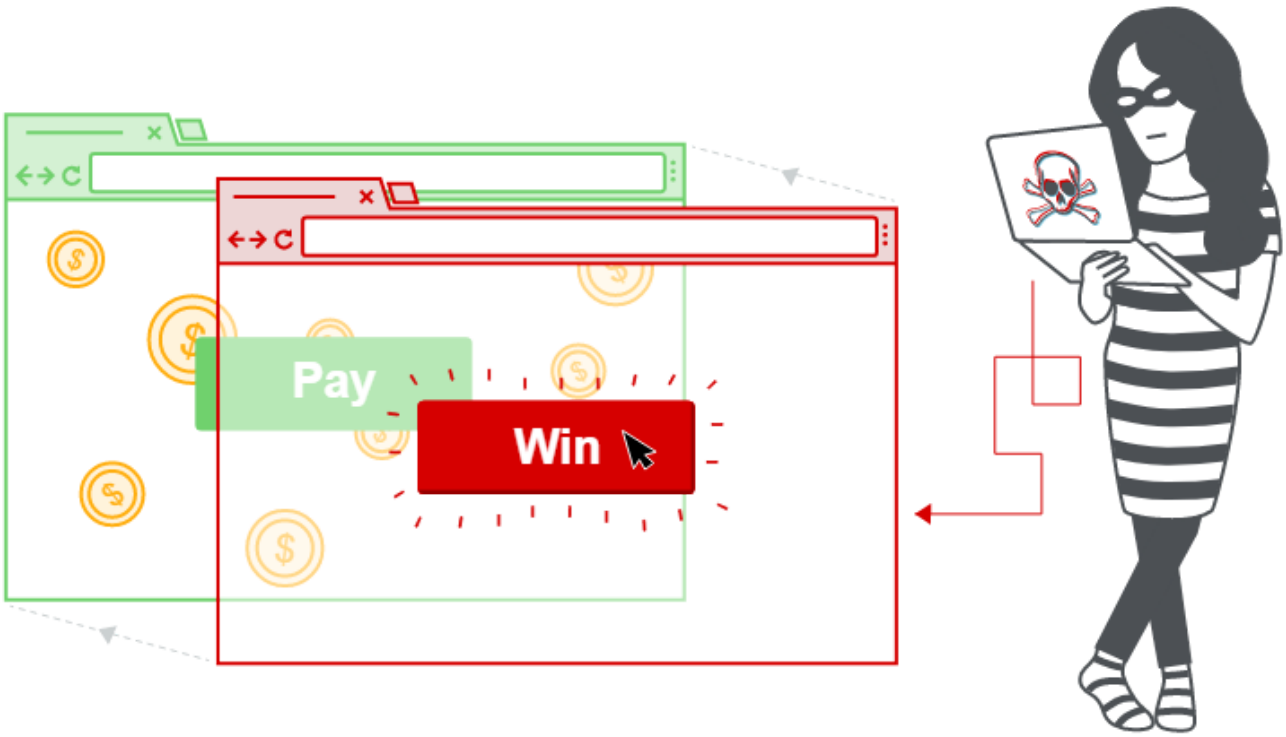
What is Clickjacking?

Clickjacking

What is Clickjacking? Tutorial & Examples - Web Security Academy

What is clickjacking?

Clickjacking, or user-interface redressing, is an attack that tricks users into clicking a malicious button that has been made to look legitimate. Attackers achieve this by using HTML page-overlay techniques to hide one web page within another.



Note : Some programs consifer ClickJacking out of scoop Vulnerability so read the Policiy berfore start Hunting.

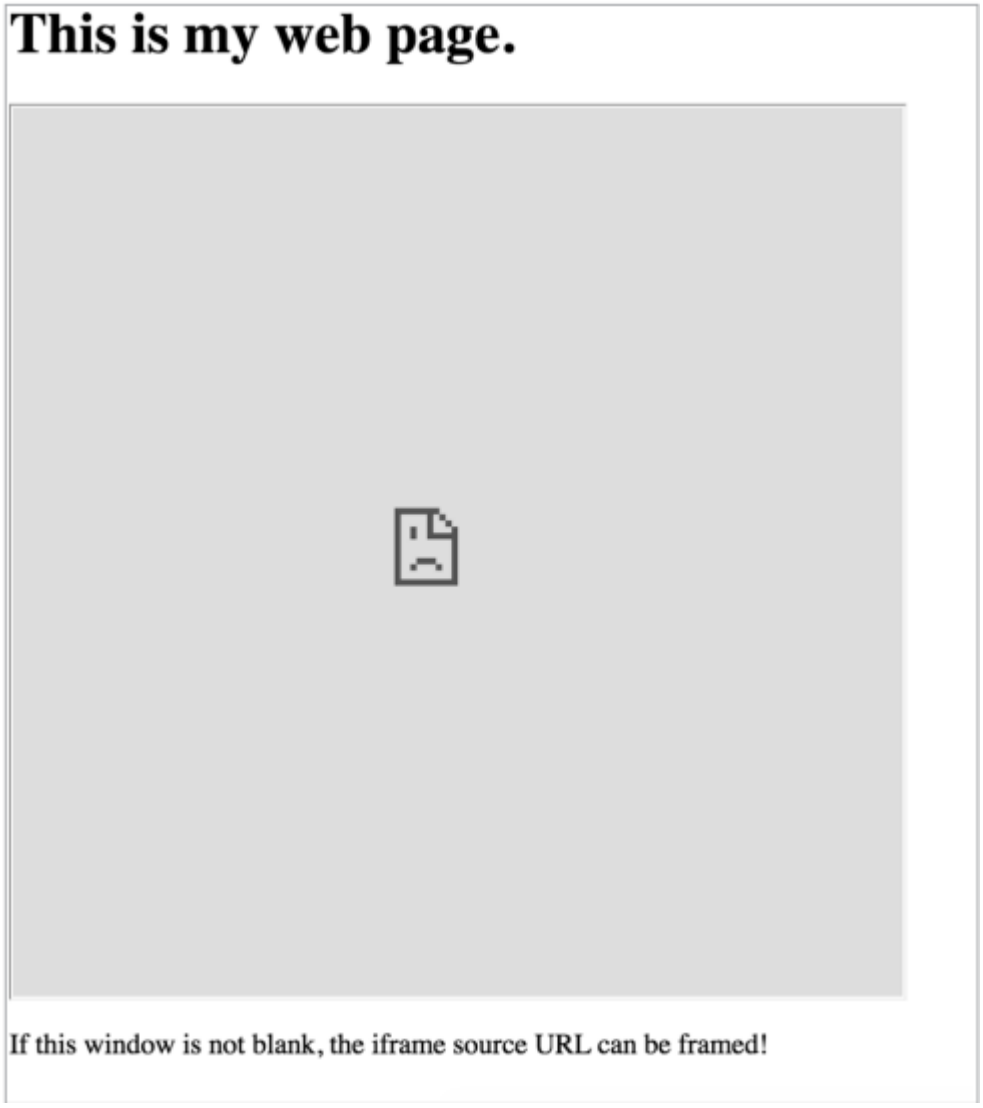
Mechanisms

Clickjacking relay on HTML feature called `iframe` . Its allow the developers to embed one page into another by placing `iframe` tag on the page.

And by placing a URL to iframe tag's attribute. Example:

```
<html>
 <h3>This is my web page.</h3>
 <iframe src="https://www.example.com" width="500" height="500"></iframe>
 <p>If this window is not blank, the iframe source URL can be framed!</p>
</html>
```

the page specified in the iframe’s src attribute can be framed!



If the iframe is blank, the iframe source cannot be framed.

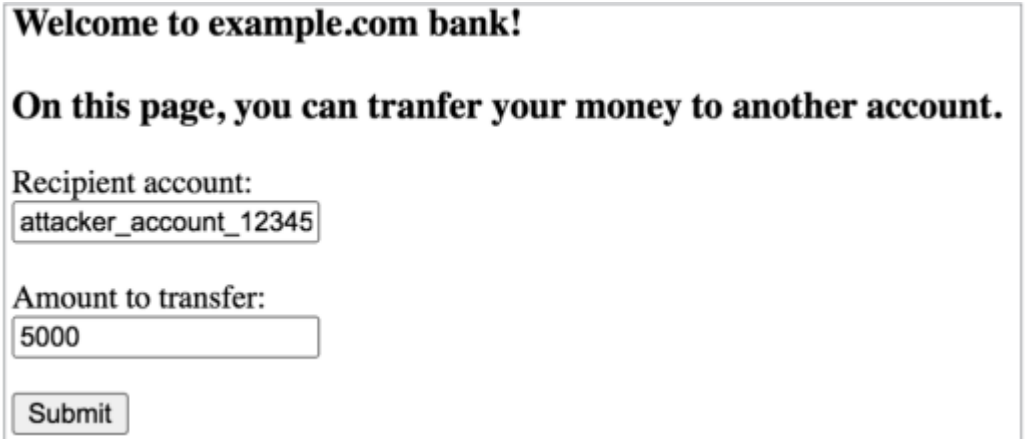
Iframes are very useful that companyies can use them to embed pre made ad. It can be used also to embed videos and audios.

For example, this iframe allows you to embed a YouTube video in an external site:

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/d1192Sqk" frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-
picture"
allowfullscreen>
</iframe>
```

Let's say that example.com is a banking site that includes a page for transferring your money with a click of a button. You can access the balance transfer page with the URL https://www.example.com/transfer_money.

This page take two things , The ID and the Amount of money transfered:



Now imagine that an attacker embeds this sensitive banking page in an iframe on their own site, like this:

```
<html>
 <h3>Welcome to my site!</h3>
 <iframe src="https://www.example.com/transfer_money?
 recipient=attacker_account_12345&amount=5000"
 width="500" height="500">
 </iframe>
        </html>
```

This iframe embeds the URL for the balance transfer page.

Take a look at this HTML page, for example:

```
<html>
 <style>
 #victim-site {
 width:500px;height:500px;
 **1** opacity:0.00001; //We then make the iframe invisible
 2 z-index:1;
 }
 #decoy {
 3 position:absolute;width:500px;height:500px; //here we make the embeded
page with the real one
 4 z-index:-1;
 }
 </style>
 <div id="decoy">
 <h3>Welcome to my site!</h3>
 <h3>This is a cybersecurity newsletter that focuses on bug bounty news and
write-ups! Please subscribe to my newsletter below to receive new
cybersecurity articles in your email inbox!</h3>
 <form action="/subscribe" method="post">
 <label for="email">Email:</label>
 5 <br>
```

```
 <input type="text" id="email" value="Please enter your email!">
 6 <br><br>
 <input type="submit" value="Submit">
 </form>
 </div>
 <iframe id="victim-site"
 src="https://www.example.com/transfer_money?
 recipient=attacker_account_12345&amount=5000"
 width="500" height="500">
 </iframe>
</html>
```

The z-index sets the stack order of different HTML elements. If two HTML elements overlap, the one with the highest z-index will be on top.

The embeded iframe page without CSS will be like :

And With CSS will be :

If the user logged in the bank site he will be loggin into the iframe page too, the request that made by the iframe will be succeed and the transacion will be succedd to.

Hunting for Clickjacking

> Find clickjacking vulnerabilities by looking for pages on the target site that contain sensitive state-changing actions and can be framed.

Step 1: Look for State-Changing Actions

It's a real valuable vulnerability only when page contain `state-changin actions`

You should look for pages that allow users to make changes to their accounts, like changing their account details or settings.

Example

If we testing on a example.com and this handling banking functionalities on bank.example.com **GO THROUGH ALL OF THE BANK FUNCTIONS AND WRITE DOWN THE STATE CHANGING OPTIONS**

State-changing requests on bank.example.com • Change password: bank.example.com/password_change • Transfer balance: bank.example.com/transfer_money • Unlink external account: bank.example.com/unlink

You should also check that the action can be achieved via clicks alone. Not using Social Engineering.

For example,

on this banking page, if the application requires users to explicitly type the recipient account and transfer amount instead of loading them from a URL parameter, attacking it with clickjacking would not be feasible.

Step 2: Check the Response Headers

Then go through each of the `state-changing functionalities` you've found and revisit the pages that contain them.

Turn on the Intercept and see the `responses` of that web page.

See if the page is being served with the `X-Frame-Options` or `Content-Security-Policy` header.

If the page is served without any of these headers, it may be `vulnerable to clickjacking`. And if the state-changing action requires users to be `logged in` when it is executed, you should also check if the site uses `SameSite cookies`. If it does, you `won't be able to exploit` a clickjacking attack on the site's features that require authentication.

You can confirm that a page is `frameable` by creating an HTML page that frames the target page. If the target page `shows up` in the frame, the page is `frameable`. This piece of HTML code is a good template:

```
<HTML>
 <head>
```

```
 <title>Clickjack test page</title>
 </head>
 <body>
 <p>Web page is vulnerable to clickjacking if the iframe is populated with
the target page!</p>
 <iframe src="URL_OF_TARGET_PAGE" width="500" height="500"></iframe>
 </body>
</html>
```

Step 3: Confirm the Vulnerability

If you can trigger the action via clicks alone through the iframe, the action is vulnerable to clickjacking.

Bypassing Protections

If the website itself fails to implement complete clickjacking protections, you might be able to bypass the mitigations.

How we can Bypass the `Fram-busting technique` if the page using it instead of SameSite Cookie.

If the top frame has the same origin as the framed page, developers may allow it, because they deem the framing site's domain to be safe.

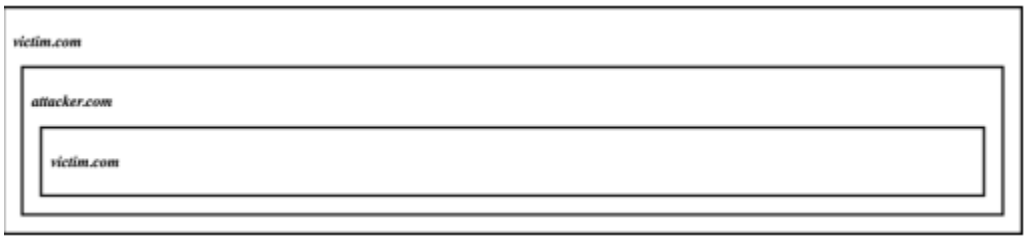
Essentially, the protection's code has this structure:

```
if (top.location == self.location){
 // Allow framing.
}
else{
 // Disallow framing.
}
```

If that is the case, search for a location on the victim site that allows you to embed custom iframes.

Such that on the social media sites : Its allow users to share links on their profile , it work by embedding the URL in an iframe to display information and a thumbnail of the link.

If you find one of these features, you might be able to bypass clickjacking protection by using the `double iframe trick`.



You can try to place your site in an iframe hosted by the victim site to bypass improper frame checking.

This way, both `top.location` and `self.location` point to victim.com. The frame-busting code would determine that the innermost victim.com page is framed by another victim.com page within its domain, and therefore deem the framing safe.

Example:

Twitter disclosed on HackerOne: Twitter Periscope Clickjacking…

The site was using the `X-Frame-Options ALLOW-FROM` directive to prevent clickjacking. This directive lets pages specify the URLs that are allowed to frame it, but it's an obsolete directive that isn't supported by `many browsers`. This means that all features on the subdomains

`https://[canary-web.pscp.tv](http://canary-web.pscp.tv/)` and `https://canary-web.periscope.tv` were vulnerable to clickjacking if the victim was using a browser that didn't support the directive, such as the latest Chrome, Firefox, and Safari browsers. Since Periscope's account settings page allows users to `deactivate` their accounts, an attacker could, for example, frame the settings page and trick users into deactivating their accounts.

Escalating the Attack

Focus on the application's most critical functionalities to achieve maximum business impact.

If we say that there are two pages that has two buttons

1. change user theme
2. transfer money to another user

Which is the Most critical ???! So you must focus on the Critical ones.

Let's say that bank.example.com contains multiple clickjacking vulnerabilities. One of them allows attackers to change an account's billing email, and another one allows attackers to send an account summary to its billing email. The malicious page's HTML looks like this:

```
<html>
 <h3>Welcome to my site!</h3>
 <iframe
 src="https://bank.example.com/change_billing_email?
email=attacker@attacker.com"
 width="500" height="500">
```

```
 </iframe>
 <iframe src="https://bank.example.com/send_summary" width="500"
height="500">
 </iframe>
</html>
```

You could first change the victim's billing email to your own email, then make the victim send an account summary to your email address to leak the information contained in the account summary report.

The most effective location in which to place the hidden button is directly on top of a Please Accept That This Site Uses Cookies! pop-up. Users usually click this button to close the window without much thought.

Finding Your First Clickjacking Vulnerability!

Now that you know what clickjacking bugs are, how to exploit them, and how to escalate them, go find your first clickjacking vulnerability! Follow the steps described in this chapter:

1. Spot the state-changing actions on the website and keep a note of their URL locations. Mark the ones that require only mouse clicks to execute for further testing.
2. Check these pages for the X-Frame-Options, Content-Security-Policy header, and a SameSite session cookie. If you can't spot these protective features, the page might be vulnerable!
3. Craft an HTML page that frames the target page, and load that page in a browser to see if the page has been framed.
4. Confirm the vulnerability by executing a simulated clickjacking attack on your own test account.
5. Craft a sneaky way of delivering your payload to end users, and consider the larger impact of the vulnerability.
6. Draft your first clickjacking report!

Prevention

First, the vulnerable page has to have functionality that executes a state-changingaction on the user's behalf : such that changing user email or password or phone number.

Second, the vulnerable page has to allow itself to be framed by an iframe on another site.

There is a HTTP Response Header called `X-Frame-Option` Specify which web page can be `rendered` in Iframe.

By defult its pages are framable if it's not specified.

```
X-Frame-Options: DENY //it cannot be framed at all
X-Frame-Options: SAMEORIGIN //allows framing from pages of the same origin
[host - protocol - port ]
```

To prevent ClickJacking attacks on all pages that contain state-changing actions must `specify` any one of the metioned headers.

The CSP [Content Security Policy] is another way to defense against these attacks.

The header `frame-ancestors` directive allow sites whether pages and be framed

```
Content-Security-Policy: frame-ancestors 'none'; //will prevent any site
from framing the page
Content-Security-Policy: frame-ancestors 'self'; //will allow the current
site to frame the page
```

Setting frame-ancestors to a specific origin will allow that origin to frame the content.

```
Content-Security-Policy: frame-ancestors 'self' *.example.com; // allow the
site and its subdomains to be framed on this page
```

The third way of prevention is `SameSite Cookie`

For example, this header will make the client browser set the value of the cookie PHPSESSID to `UEhQU0VTU0lE`: `Set-Cookie: PHPSESSID=UEhQU0VTU0lE`

In addition to the basic cookie_name=cookie_value designation, When the SameSite flag on a cookie is set to Strict or Lax, that cookie won’t be sent in requests made within a third-party iframe:

```
Set-Cookie: PHPSESSID=UEhQU0VTU0lE; Max-Age=86400; Secure; HttpOnly;
SameSite=Strict
Set-Cookie: PHPSESSID=UEhQU0VTU0lE; Max-Age=86400; Secure; HttpOnly;
SameSite=Lax
```

This means that any clickjacking attack that requires the victim to be authenticated, like the banking example we mentioned earlier, would not work, even if no HTTP response header restricts framing, because the victim won’t be authenticated in the clickjacked request.