X-Frame-Options

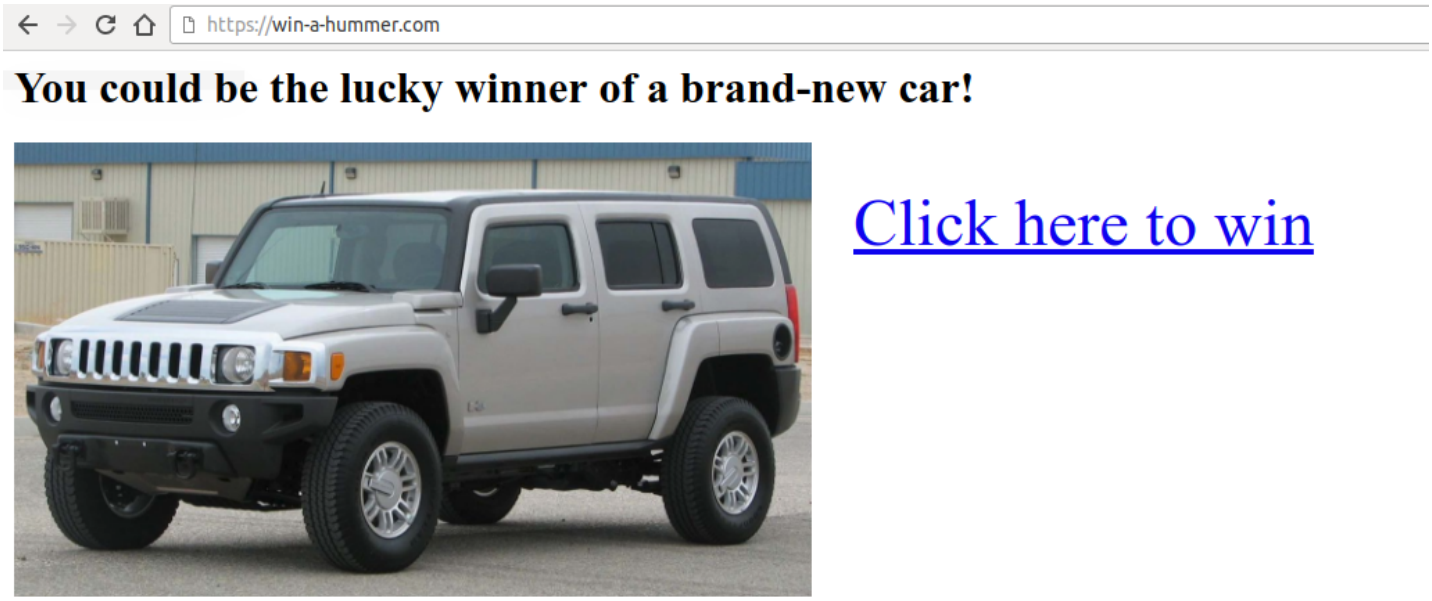
In this lesson, we'll study X-Frame-Options.

We'll cover the following

- What is clickjacking?
- A runnable example

What is clickjacking?

Imagine seeing a web page like this on your screen:



Who wouldn't want to be a proud owner of a brand-new hummer?

As soon as you click on the link, you realize that all the money in your bank account is gone. What happened?

You were a victim of a *clickjacking* attack! An attacker directed you to their website, which displays an attractive link to click. Unfortunately, they also embedded an iframe from `your-bank.com/transfer?amount=10000000&to=attacker@example.com` in the page but hid it by setting its opacity to 0%. Then, instead of clicking on the original page and winning a brand-new hummer, the browser captured a click on the iframe, a dangerous click that

confirmed the transfer of money. Most banking systems require you to specify a

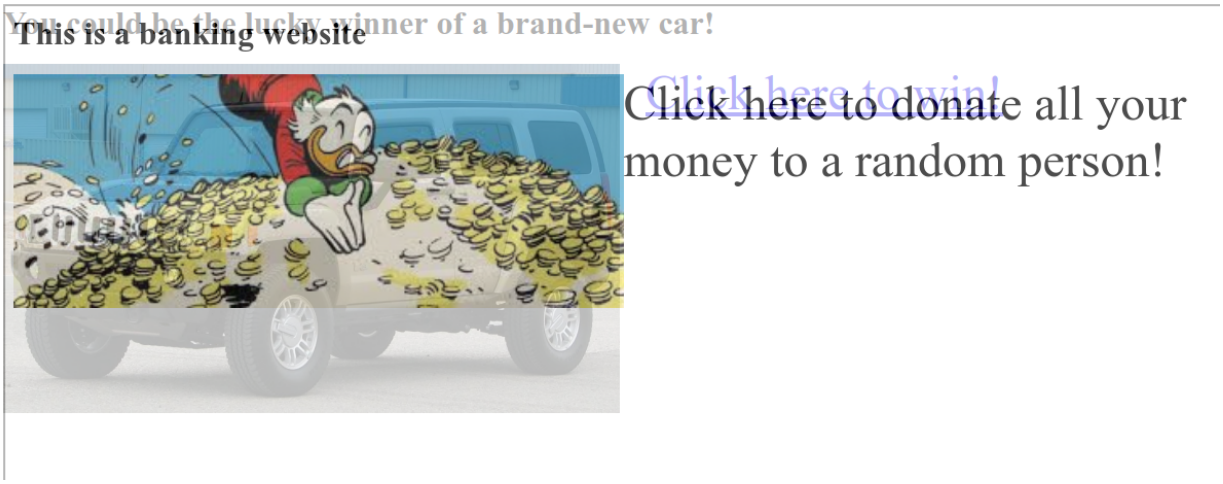
one-time PIN code to confirm transactions, but your bank hasn't caught up with the times, and all of your money is gone.

The example is pretty extreme but should help you understand the possible consequences of a clickjacking attack. The user intends to click on a particular link while the browser will trigger a click on the invisible page that's been embedded as an iframe.

A runnable example

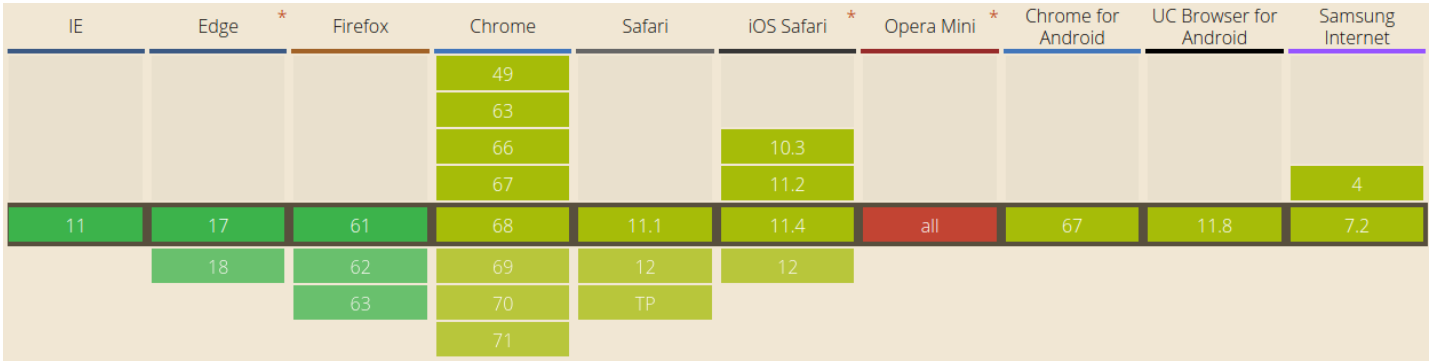
I have included an example of this vulnerability in the code below. Run it and see the website. If you run the example and try clicking on the appealing link, you will see the actual click is intercepted by the iframe, which increases its opacity so that's easier for you to spot the problem.

```
var qs = require('querystring')
var url = require('url')
var fs = require('fs')

require('http').createServer((req, res) => {
  let path = url.parse(req.url).pathname === '/' ? '/hummer.html' : url.parse(req.url).pathname
  let query = qs.parse(url.parse(req.url).query)
  let headers = {}

  if (query.xfo === "on") {
    headers['X-Frame-Options'] = "DENY"
  }

  if (query.csp === "on") {
    headers['Content-Security-Policy'] = "frame-ancestors 'none'"
  }

  res.writeHead(200, headers)
  let content = ""

  try {
    content = fs.readFileSync(__dirname + path)
    if (path.endsWith('.html')) {
      content = content.toString().replace('__QUERY__', req.url.replace('/', ''))
    }
  } catch (err) {}
  res.end(content)
}).listen(7888)
```



During a clickjacking attack, a transparent iframe is usually capturing user interactions such as clicks

Luckily, browsers have come up with a simple solution to the problem, `X-Frame-Options` (abbr. XFO). XFO lets you decide whether your app can be embedded as an iframe on external websites. Popularized by Internet Explorer 8, XFO was first introduced in 2009 and is still supported by all major browsers. When a browser sees an iframe, it loads it and verifies that its XFO allows its inclusion in the current page before rendering it.



Browser support for XFO

The supported values are:

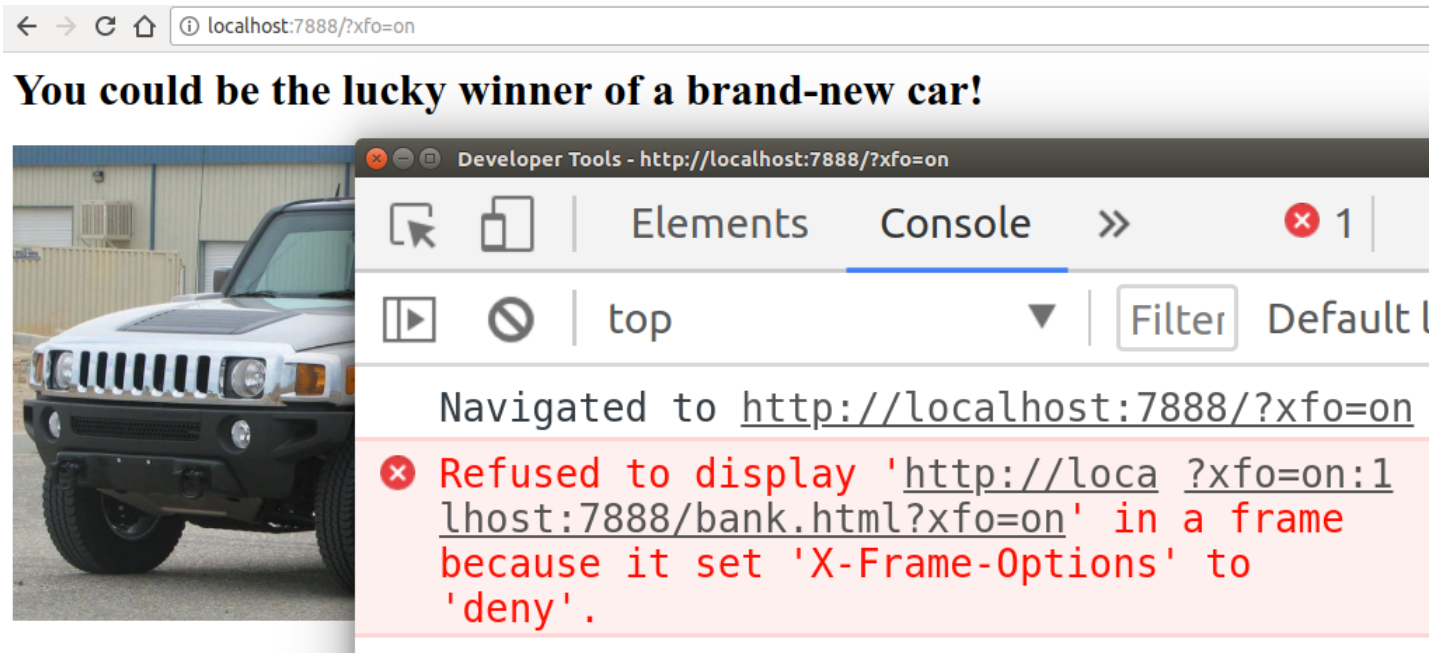
- `DENY`: This web page cannot be embedded anywhere. This is the highest level of protection as it doesn't allow anyone to embed our content.
- `SAMEORIGIN`: Only pages from the same domain as the current one can embed this page. This means that `example.com/embedder` can load `example.com/embedded` so long as its policy is set to `SAMEORIGIN`. This is a more relaxed policy that allows owners of a particular website to embed their own pages across their application.

- `ALLOW-FROM uri`: Embedding is allowed from the specified URI. We could, for example, let an external, authorized website embed our content by using `ALLOW-FROM https://external.com`. This is generally used when you intend to allow a third party to embed your content through an iframe.

An example HTTP response that includes the strictest XFO policy possible looks like this:

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Frame-Options: DENY

...
```

In order to showcase how browsers behave when XFO is enabled, we can simply change the URL of our example to `http://localhost:7888/?xfo=on`. The `xfo=on` parameter tells the server to include `X-Frame-Options: deny` in the response, and we can see how the browser restricts access to the iframe.



Clickjacking is prevented with XFO flag turned on

XFO was considered the best way to prevent frame-based clickjacking attacks until another header came into play years later, the Content Security Policy.

In the next lesson, we'll study the `Content-Security-Policy`.