Is it possible to scrape an HTML page with JavaScript from inside of a web browser?

To be perfectly honest I wasn’t sure so I decided to try it out.

Full disclaimer here, I didn’t actually succeed. However, it was a great learning experience for me and I think you guys could benefit from seeing what I did and where I went wrong. Who knows, maybe you can take what I’ve done and figure it out for yourself!

You can jump to any of these methods if you like…

|  |  |  |
| --- | --- | --- |
| [CORS](http://scraping.pro/web-scraping-with-javascript-load-html-page/#cors) | [No *Referer* header request](http://scraping.pro/web-scraping-with-javascript-load-html-page/#no_referer) | [WordPress pages’ load](http://scraping.pro/web-scraping-with-javascript-load-html-page/#wp_load) |

Let’s say you’re at mysite.com (in your browser) and want to run a script to load some data from example.com. The simplest solution for requesting content via JavaScript would be to request it through an XML Http Request (XHR):



|  |  |
| --- | --- |
| 1  2  3  4 | xmlhttp=new XMLHttpRequest();  xmlhttp.open("GET", "http://example.com/data/json", false);  xmlhttp.send();  var data = JSON.parse(xmlhttp.responseText); |

However, this pertains to the cross-origin requests (as opposed to same-origin request), which turns out to be the unbreakable wall for requesting page content from pure JavaScript.

Same Origin Policy

Same-origin policy restricts how a document or script loaded from one origin can interact with a resource from another origin. It is a critical security mechanism for isolating potentially malicious documents. [Source](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy" \t "_blank).

Same-origin policy requires that in web requests made from the client side (mysite.com) both domain name and protocol (http, https) should be equal. So security limitations do not allow you to request another domain website (example.com) or any of its resources.

Can you bypass it? No.

You might think you can just bypass this. In order to request foreign domain resources we imitate the same domain origin. How? By obfuscating *Referer* header in XML Http Request, see the following:



|  |  |
| --- | --- |
| 1 | xmlhttp.setRequestHeader("Referer","http://domain\_for\_scrape.com"); |

This is **explicitly disallowed** by the [XHR specification](http://www.w3.org/TR/XMLHttpRequest/#dom-xmlhttprequest-setrequestheader). So in JavaScript XHR you can’t explicitly set up an arbitrary header.

Resources allowed to be requested from a foreign domain

Now, there are services that do allow cross-origin resource sharing. This is applicable for distributed resource sharing to diminish a server resource load. Eg. CSS stylesheets, images, and scripts might be served from foreign domain servers. Here are some examples of resources which may be embedded cross-origin:

* JavaScript with <script src="..."></script>. Error messages for syntax errors are only available for same-origin scripts.
* CSS with <link rel="stylesheet" href="...">.
* Images with <img>. Supported image formats include PNG, JPEG, GIF, BMP, SVG.
* Media files with <video> and <audio>.
* Plug-ins with <object>, <embed> and <applet>.
* Fonts with @font-face. Some browsers allow cross-origin fonts, others require same-origin fonts.
* Anything with <frame> and <iframe>. A site can use the X-Frame-Options header to prevent this form of cross-origin interaction.

One example is the jQuery code which is often served from [ajax.googleapis.com](http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js) domain:



|  |  |
| --- | --- |
| 1 | <script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script> |

But, to the developers’ joy the *Cross-Origin Resource Sharing policy* has been recently (January 2014) introduced. This allows to resolve the Same-Origin Policy restriction.

Cross-Origin Resource Sharing (CORS)

The main concept is that a target server may allow some other origins (or all of them) to request its resources. Server configured for allowing cross-origin requests is useful for the cross-domain API access of its resources.

If a server allows CORS it’ll respond with **Access-Control-Allow-Origin:\*** header.



|  |  |
| --- | --- |
| 1 | Access-Control-Allow-Origin: \* |

If a resource owner’s restricts the sharing with only a certain domain, the server will respond with:



|  |  |
| --- | --- |
| 1 | Access-Control-Allow-Origin: http://mysite.com |

You might do a [preflight request](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing" \l "Preflight_example" \t "_blank) to make clear if a server allows foreign domain access.

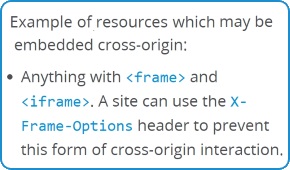
 Cross-Origin Resource Sharing (CORS) is a W3C spec that allows cross-domain communication from web browser.

Read more of CORS [here](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS" \t "_blank).  
How to set an access control in Apach server (enabling CORS) see [here](https://developer.mozilla.org/en-US/docs/Web/HTTP/Server-Side_Access_Control" \t "_blank).  
[CORS tester](http://client.cors-api.appspot.com/client#?client_method=GET&client_credentials=false&client_headers=User-Agent%3AMozilla%2F775.0%20(Windows%20NT%206.3%3B%20WOW64)%20AppleWebKit%2F537.36%20(KHTML%2C%20like%20Gecko)%20Chrome%2F46.0.2490.80%20Safari%2F537.36&server_url=http%3A%2F%2Ftarex.ru&server_enable=true&server_status=200&server_credentials=false&server_tabs=local).

Wrap up

Eventually, site owners will allow CORS only for the API access since it’s unlikely they will make their private web data cross-origin accessable. The attempt to scrape other sites’ content with JavaScript provides a very limited scope.

No Referer form submission

[](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)

*Same Origin Policy exclusion.*

We’ve mentioned before that <iframe> loading foreing data in it works by neglecting same-domain policy.

Let’s try to use the form submission with no *Referer* header. Most of the sites approve the request if *Referer* header is empty (omitted). Websites do this because they don’t want to lose sort of 1% of their traffic. So we make a simple procedure that is called for a chosen domain with requesting thru virtual form submission:

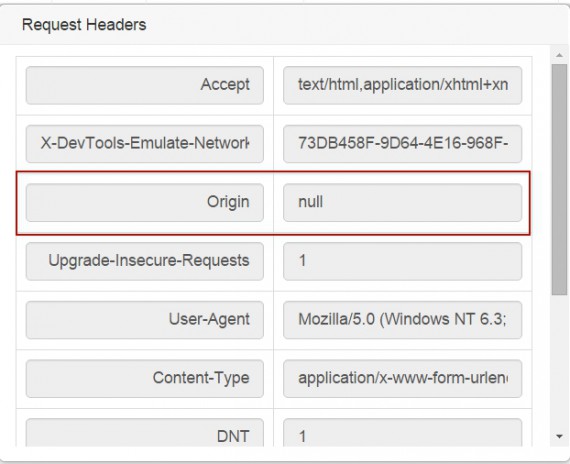
JavaScript

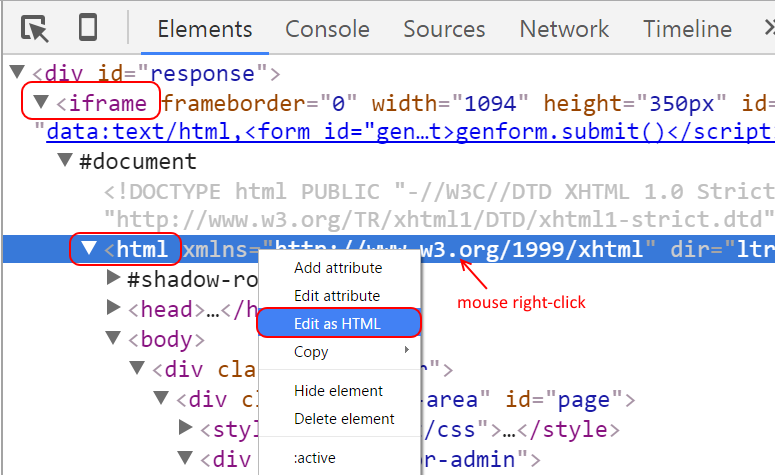


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | function noref\_request(site\_url){     var virtualForm = 'data:text/html,<form id="genform" action="http://www.' +site\_url + '" method="GET"> <input type="submit" ></form> <script>genform.submit()<\/script>';       var iframe = document.createElement('iframe');     iframe.frameBorder=0;     iframe.width= window.innerWidth;     iframe.height= "350px";     iframe.seamless='seamless';     iframe.setAttribute("src", virtualForm);     document.getElementById("response").appendChild(iframe);  } |

This code, when called client-side, adds new <iframe> into a web page and loads needed resource into a browser page. The whole code is [here](http://codeshare.io/GAN8B" \t "_blank). Kind of loading.

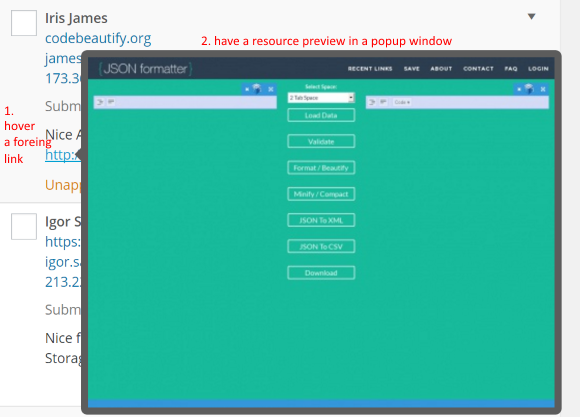
See the following web sniffer’s shot showing the *Origin* header being *null* and no *Referer*header present.

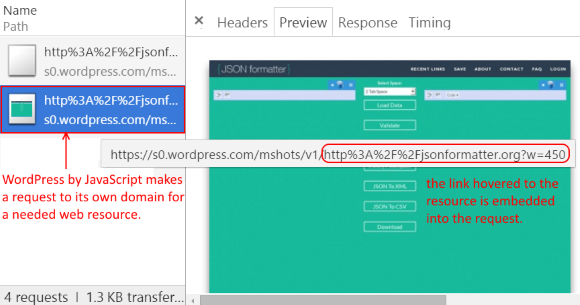


So basically you might load a foreign page into your browser page by JavaScript. But still the Same Origin Policy, applied in all major browsers, **forbids access to the fetched HTML**. Cross-origin contents can not be read by JavaScript. No major browser will allow that to secure against XSS attacks. Surprisingly, you can watch the cross-site request response HTML code thru browser’s web developer tools (F12, [an example of using them](http://scraping.pro/find-xpath-using-web-developer-tools/" \l "picture" \t "_blank)) and manually copy/paste it:  
  
The loaded site will seamlessly work in an iframe, yet, you can’t have an access to its HTML. You can get the page’s screenshot as an image, but it’s not sufficient for full-scale web scraping.

How does WordPress load foreign page shots into its admin panel

WordPress CMS can load of foreign resources with server-side call (if having access to wp-admin – just visit: <site>/wp-admin/edit-comments.php). If the user hovers a comment website link, the CMS’ JavaScript automatically invokes a request to the WordPress home server:



Now the CMS makes HTTP request to its own server, embedding the link to the foreign resource. Obviously the WordPress server makes request to the resource by provided link of interest and returns the content:  


The only the thing is that the content returned by WordPress being an image: *content-type: image/jpeg.* You can program server to return HTML code, but that’s server-side data extraction.

Conclusion

The client-side (from your browser) scraping with JavaScript is not practical today. (1) The browser capabilities are far less compared to web servers (speed, memory, etc.). (2) The Same-Origin Policy safeguards sites from cross-origin requests, avoiding XSS attacks threat. CORS is limited scope applicable. (3) I’ve also tried to emulate the cross-domain HTTP request by a virtual form submission to load a result into an iframe, but this failed since the browser restrictions forbid scripts to handle raw response HTML cause of XSS attacks threat. (4) The last option is the indirect requesting thru a domain server (mysite.com, who actually extracts). Eg. WordPress loading foreign pages’ previews.

Feel free to add more to this topic (using comments).