This popular article was updated on 23rd June, 2016 to address quality issues. Comments pertaining to the old article were removed.

If you're developing a web-based application and are trying to load data from a domain which is not under your control, the chances are that you've seen the following message in your browser's console:

XMLHttpRequest cannot load http://external-domain/service. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://my-domain' is therefore not allowed access.

In this article, we'll look at what causes this error and how we can get around it by using jQuery and JSONP to make a cross-domain Ajax call.

Same-origin Policy

Regular web pages can use the XMLHttpRequest object to send and receive data from remote servers, however they're restricted in what they can do by the same origin-policy. This is an important concept in the browser security model and dictates that a web browser may only allow scripts on page A to access data on page B if these two pages have the same origin. The origin of a page is defined by its *protocol*, *host* and *port number*. For example the origin of this page is 'https', 'www.sitepoint.com', '80'.

The same-origin policy is a saftey mechanism. It prevents scripts from reading data from your domain and sending it to their servers. If we didn't have this, it would be easy for a malicious website to grab your session information to another site (such as Gmail or Twitter) and execute actions on your behalf. Unfortunately, it also causes the error we see above and often poses a headache for developers trying to accomplish a legitimate task.

A failing example

Let's look at what doesn't work. Here's a JSON file residing on a different domain which we would like to load using jQuery's getJSON method.

```
$.getJSON(
```

```
    "http://run.plnkr.co/plunks/v8xyYN64V4nqCshgjKms/data-1.json",
    function(json) { console.log(json); }
);
```

If you try that out in your browser with an open console, you will see a message similar to the one above. So what can we do?

A Possible Workaround

Luckily, not everything is affected by the same-origin policy. For example, it is quite possible to load an image or a script from a different domain into your page—this is exactly what you are doing when you include jQuery (for example) from a CDN.

This means that we are able to create a `<script>` tag, set the `src` attribute to that of our JSON file and inject it into the page.

```
var script = $("<script />", {
    src: "http://run.plnkr.co/plunks/v8xyYN64V4nqCshgjKms/data-1.json",
    type: "application/json"
  }
);

$("head").append(script);
```

Although that works, it doesn't help us much, as we have no way of getting at the data it contains.

Enter JSONP

JSONP (which stands for JSON with Padding) builds on this technique and provides us with a way to access the returned data. It does this by having the server return JSON data wrapped in a function call (the "padding") which can then be interpreted by the browser. This function must be defined in the page evaluating the JSONP response.

Let's see what that would look like with our previous example. Here's an updated JSON file which wraps the original JSON data in a `jsonCallback` function.

```
function jsonCallback(json){
  console.log(json);
```

```
}

$.ajax({

  url: "http://run.plnkr.co/plunks/v8xyYN64V4nqCshgjKms/data-2.json",
  dataType: "jsonp"
});
```

This logs the expected result to the console. We now have (albeit rather limited) cross-domain Ajax.

3rd Party APIs

Some 3rd party APIs let you specify a name for the callback function which should be executed when the request returns. One such API is the GitHub API.

In the following example we're getting the user information for John Resig (jQuery creator) and using a `logResults` callback function to log the response to the console.

```
function logResults(json){
  console.log(json);
}

$.ajax({
  url: "https://api.github.com/users/jeresig",
  dataType: "jsonp",
  jsonpCallback: "logResults"
});
```

This can also be written as:

```
$.getJSON("https://api.github.com/users/jeresig?callback=?",function(json){
  console.log(json);
});
```

The `?` on the end of the URL tells jQuery that it's dealing with a JSONP request instead of JSON. jQuery then automatically registers the callback function which it calls when the request retruns.

If you'd like to learn more about jQuery's `getJSON` method, check out: Ajax/jQuery.getJSON Simple Example

Caveats

But as you might have realized by now, there are some drawbacks to this approach.

For example, JSONP can only perform cross-domain GET requests and the server must explicitly support it. JSONP is also not without its security concerns, so let's briefly look at some other solutions.

Using a proxy

Server-side code is not bound by the same origin policy and can perform cross-origin requests without a problem. You could therefore make some kind of proxy and use that to retrieve whatever data you need. With reference to our first example:

```
/* proxy.php */
$url = "http://run.plnkr.co/plunks/v8xyYN64V4nqCshgjKms/data-1.json";
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$result = curl_exec ($ch);
curl_close ($ch);
echo $result;
```

And on the client side:

```
$.getJSON("http://my-domain.com/proxy.php", function(json) {
  console.log(json);
})
```

But this approach also has its downsides. For example, if the third-party site uses cookies for authentication, this will not work.

CORS

Cross-Origin Resource Sharing (CORS) is a W3C spec to allow cross-domain communication from the browser. This is done by including a new `Access-Control-Allow-Origin` HTTP header in the response.

With reference to our first example, you could add the following to a `.htaccess` file (assumes Apache) to permit requests from a different origins:

```
Header add Access-Control-Allow-Origin "http://my-domain.com"
```

(*If you have a server running something other than Apache, look here: http://enable-cors.org/server.html*)

You can find out more about CORS in one of our recent tutorials: An In-depth Look at CORS

Conclusion

JSONP allows you to sidestep the same-origin policy and to some extent make cross-domain Ajax calls. It's not a silver bullet, and it certainly has its issues, but in some cases it can prove invaluable when fetching data from a different origin.

JSONP also makes it possible to pull in a variety of content from different services. Many prominent sites provide JSONP services (for example Flickr), allowing you access to their content via a predefined API. You can find a comprehensive list of them in the ProgrammableWeb API directory.