Deal with CORS error in Chromium

Dealing with the CORS error when loading images in Chromium, Chrome or Edge.

By Kamen Kotsev

5 min. read View original

Have you ever had to load images in JavaScript using the CORS Header `crossOrigin="Anonymous"`?

In a recent project of ours, we've encountered an issue when fetching images with CORS headers in JavaScript. The error messages stated:

> Access to image at '${url}' from origin '${origin}' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

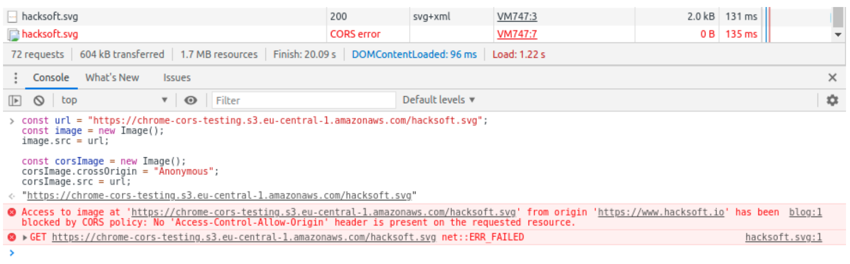
Reproducing the error

TL;DR:

If you open a Google Chrome/Chromium/Microsoft Edge browser. Start the console in a random website (for example this one) and run this code in it - you should be able to see the error in the network tab.

```
const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
const image = new Image();
image.src = url;

const corsImage = new Image();
corsImage.crossOrigin = "Anonymous";
corsImage.src = url;
```

You can see in the network tab, that the first image, called without setting crossOrigin, loaded correctly, and the second image, called with crossOrigin="Anonymous" has an error.



hacksoft.svg	200	svg+xml	VM747:3	2.0 kB	131 ms
hacksoft.svg	CORS error		VM747:7	0 B	135 ms

72 requests | 604 kB transferred | 1.7 MB resources | Finish: 20.09 s | DOMContentLoaded: 96 ms | Load: 1.22 s

```
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
Access to image at 'https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg' from origin 'https://www.hacksoft.io' has been  blog:1
blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
▸GET https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg net::ERR_FAILED                hacksoft.svg:1
>
```

Long version:

The steps to reproduce the issue are the following:

- Open a browser running on the Chromium core. The most widely used of those are Chromium, Google Chrome and Microsoft Edge. I'm going to use Google Chrome to demonstrate it.
- Open a random website. For example - this one.
- Open the console in your browser devtools.
- Chose an image url from a different host that has CORS specifications. It's important to be from a different host, and to not return the

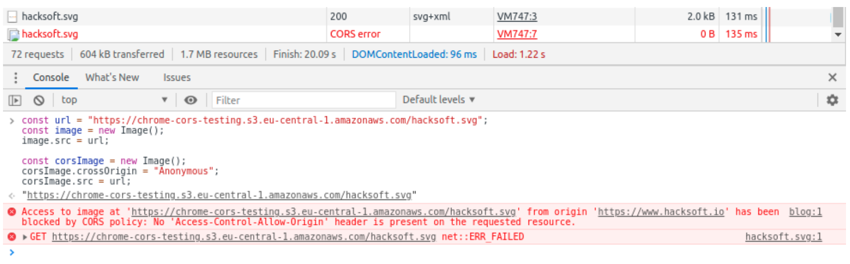
`Access-Control-Allow-Origin: *` header, so we can trigger the CORS check. This happens for almost all of the s3-hosted images. The example that I have is this url: https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg
- Load the image using JavaScript:

```
const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
const image = new Image();
image.src = url;
```

- Load the image again, but this time add a `Access-Control-Allow-Origin: *` header. Do this by adding the image property `.crossOrigin = "Anonymous"`

```
const corsImage = new Image();
corsImage.crossOrigin = "Anonymous";
corsImage.src = url;
```

The result should look something like this:



```
hacksoft.svg      200          svg+xml    VM747:3    2.0 kB   131 ms
hacksoft.svg      CORS error              VM747:7    0 B      135 ms
72 requests | 604 kB transferred | 1.7 MB resources | Finish: 20.09 s | DOMContentLoaded: 96 ms | Load: 1.22 s
```

```
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
✖ Access to image at 'https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg' from origin 'https://www.hacksoft.io' has been  blog:1
  blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ ▶GET https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg net::ERR_FAILED                        hacksoft.svg:1
>
```

Note that the second time we try to load the image - Chrome returns a CORS error instead of a response object.

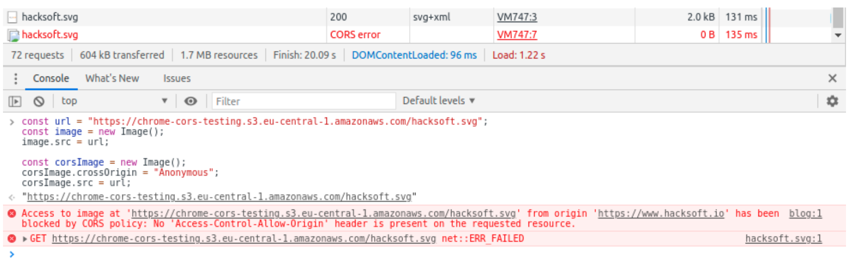
The reason

So why does Google Chrome throw an error when the url is accessed with a CORS header? Well, first, you should know why do websites use the CORS policy. There is a very good article explaining this.

Let's explore how does the browser fetch images and resources.
It sends a GET request for the image with certain headers. It then downloads the image and then caches it for further use.
Before loading any image, it checks the cache first, to see if it already downloaded it at some point. If it finds the image there - the browser doesn't send a GET request for the image, but rather just takes it from the cache and serves it back to you. This saves load time and network data when you often visit the same website.

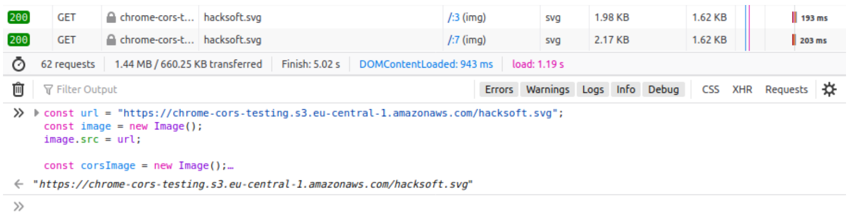
The issue that we have here, is related to Chromium's way of caching images, and it doesn't appear to happen in browsers based on different engines:

Chrome:


```
hacksoft.svg    200          svg+xml    VM747:3    2.0 kB   131 ms
hacksoft.svg    CORS error              VM747:7      0 B    135 ms
72 requests | 604 kB transferred | 1.7 MB resources | Finish: 20.09 s | DOMContentLoaded: 96 ms | Load: 1.22 s
```

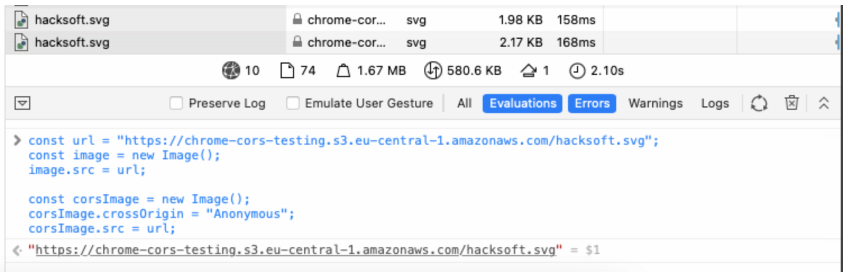
```
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
✖ Access to image at 'https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg' from origin 'https://www.hacksoft.io' has been  blog:1
  blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ ▶GET https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg net::ERR_FAILED                              hacksoft.svg:1
>
```

Firefox:


```
200  GET  chrome-cors-t...  hacksoft.svg  /:3 (img)  svg  1.98 KB  1.62 KB  193 ms
200  GET  chrome-cors-t...  hacksoft.svg  /:7 (img)  svg  2.17 KB  1.62 KB  203 ms
62 requests | 1.44 MB / 660.25 KB transferred | Finish: 5.02 s | DOMContentLoaded: 943 ms | load: 1.19 s
```

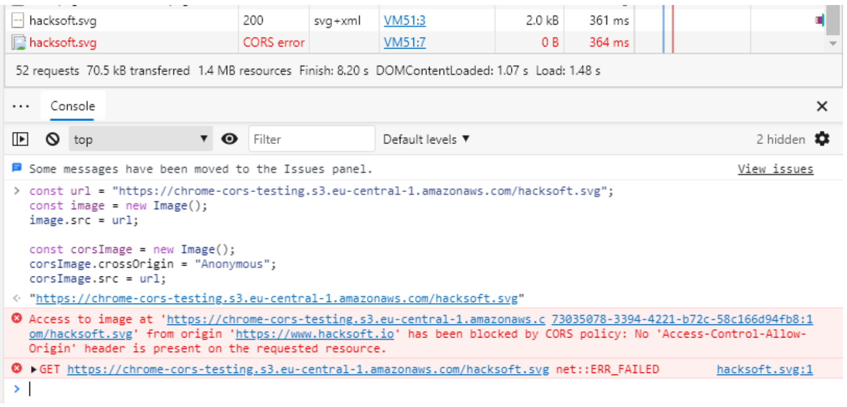
```
>> ▶ const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
     const image = new Image();
     image.src = url;

     const corsImage = new Image();…
← "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
>>
```

Safari:



```
hacksoft.svg    chrome-cor...   svg   1.98 KB   158ms
hacksoft.svg    chrome-cor...   svg   2.17 KB   168ms

10   74   1.67 MB   580.6 KB   1   2.10s

Preserve Log   Emulate User Gesture   All   Evaluations   Errors   Warnings   Logs

> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg" = $1
```

Edge:



```
hacksoft.svg   200          svg+xml   VM51:3   2.0 kB   361 ms
hacksoft.svg   CORS error             VM51:7   0 B      364 ms
52 requests  70.5 kB transferred  1.4 MB resources  Finish: 8.20 s  DOMContentLoaded: 1.07 s  Load: 1.48 s

Console
top   Filter   Default levels   2 hidden
Some messages have been moved to the Issues panel.                                    View issues
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
Access to image at 'https://chrome-cors-testing.s3.eu-central-1.amazonaws.c 73035078-3394-4221-b72c-58c166d94fb8:1
om/hacksoft.svg' from origin 'https://www.hacksoft.io' has been blocked by CORS policy: No 'Access-Control-Allow-
Origin' header is present on the requested resource.
▸GET https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg net::ERR_FAILED          hacksoft.svg:1
> |
```

> Note that Microsoft Edge also has this issue, because it's based on the Chromium engine.

The issue comes from the way that Chromium caches the images. The way that the initial image is cached is - without the CORS headers. So next time when we want to fetch the image, with CORS headers - Chromium attempts to serve the image from the cache.
The issue is that the image didn't have the CORS headers when we first fetched it (which could happen when you browse through the website and see the image rendered in an `<img>` tag).
And since the image didn't have the CORS headers initially, and has them now - Chromium returns a CORS error.
It's a well known issue in Chromium and has been described in the chromium bug tracking software:

https://bugs.chromium.org/p/chromium/issues/detail?id=409090

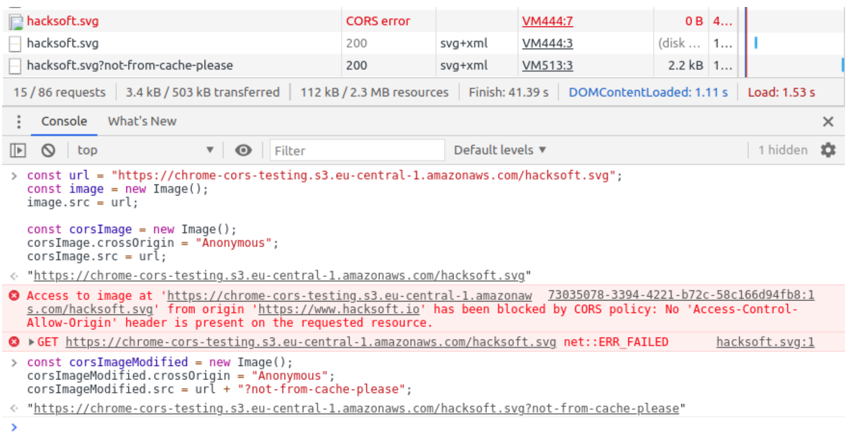
The developer team working on Chromium however flagged the issue as `WontFix(Closed)` Because this is likely the intended behavior of the Chromium engine.

Solution

In order to solve this issue, we can simply add a dummy GET parameter in the url when fetching the required image. This will force the browser to not use the cached image from before, but to send a new GET request for the image because the URL is now different from the one that Chromium has cached.

```
const corsImageModified = new Image();
corsImageModified.crossOrigin = "Anonymous";
corsImageModified.src = url + "?not-from-cache-please";
```

Example:



hacksoft.svg	CORS error		VM444:7	0 B	4...
hacksoft.svg	200	svg+xml	VM444:3	(disk ...	1...
hacksoft.svg?not-from-cache-please	200	svg+xml	VM513:3	2.2 kB	1...

15 / 86 requests | 3.4 kB / 503 kB transferred | 112 kB / 2.3 MB resources | Finish: 41.39 s | DOMContentLoaded: 1.11 s | Load: 1.53 s

```
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;

  const corsImage = new Image();
  corsImage.crossOrigin = "Anonymous";
  corsImage.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
✖ Access to image at 'https://chrome-cors-testing.s3.eu-central-1.amazonaw   73035078-3394-4221-b72c-58c166d94fb8:1
  s.com/hacksoft.svg' from origin 'https://www.hacksoft.io' has been blocked by CORS policy: No 'Access-Control-
  Allow-Origin' header is present on the requested resource.
✖ ▶GET https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg net::ERR_FAILED      hacksoft.svg:1
> const corsImageModified = new Image();
  corsImageModified.crossOrigin = "Anonymous";
  corsImageModified.src = url + "?not-from-cache-please";
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg?not-from-cache-please"
>
```

> Note that here the image loaded correctly, and we just added one dummy GET parameter

The GET parameter you add doesn't matter, as long as the resulting URL is different than the initial (cached) image URL.
By just adding a dummy GET parameter, you will get the same image that you need, but this time Chromium will send a new request for it, containing the CORS headers in it.

Another upside of this solution is that it doesn't bother all of the other browsers as well. So it will fix the error that your users are getting in Chrome, Edge and Chromium, without affecting the experience that all of your other users are having.

When is this useful?

"But, hey, when will I need to fetch the same image with different headers?" you ask.
That's a good question. And an example use case would be - when rendering that image in a canvas that you need to scrape later.

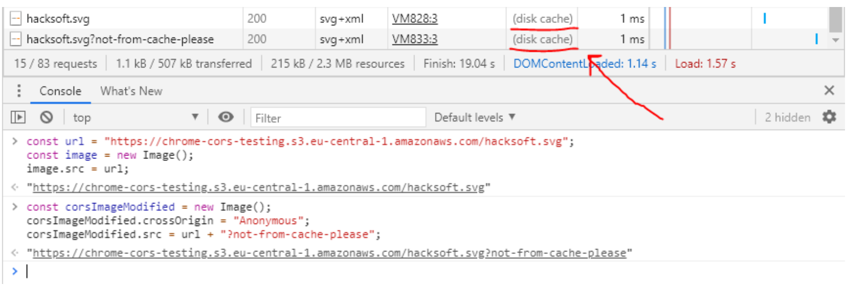
You see, when you render an image in a canvas, it becomes tainted. This is a security feature that stops you from reading what's in the canvas after you've added that image.
In order to render an image, and use the information from the canvas later - the image should be loaded with the `Access-Control-Allow-Origin` header.

This is all well and good, but if that image was shown in an <img> tag before the user got to see it in the canvas - then Chrome cached it, and you hit the exact same issue that this article solves.

Note about caching

If we want to cache the image with the CORS header, we can always use the same `dummy GET parameter` when we call the image url. Chromium will cache it with that "different" url that we created, and will use it when we call it next time without raising the error.

Example:



```
hacksoft.svg                          200   svg+xml   VM828:3   (disk cache)   1 ms
hacksoft.svg?not-from-cache-please    200   svg+xml   VM833:3   (disk cache)   1 ms
15 / 83 requests | 1.1 kB / 507 kB transferred | 215 kB / 2.3 MB resources | Finish: 19.04 s | DOMContentLoaded: 1.14 s | Load: 1.57 s
```

```
> const url = "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg";
  const image = new Image();
  image.src = url;
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg"
> const corsImageModified = new Image();
  corsImageModified.crossOrigin = "Anonymous";
  corsImageModified.src = url + "?not-from-cache-please";
< "https://chrome-cors-testing.s3.eu-central-1.amazonaws.com/hacksoft.svg?not-from-cache-please"
> |
```

> Note here, that the browser takes both the image without CORS and with CORS from the cache, because they were cached before.

TL;DR

Problem:

If you've loaded an image in Chrome, Edge, Chromium or other Chromium-based browser, and the browser cached that image. When you call for that same image with the `Access-Controll-Allow-Origin` header (or `crossOrigin="Anonymous"` if you're doing it in JavaScript) - Chromium returns an error response because the initially cached image didn't have that header.

Solution:

When calling the image url with the `crossOrigin="Anonymous"` header, add a dummy GET parameter at the end of the URL. This will essentially change the resource, so Chrome won't look into the cache and will call the "new" url instead, giving you the image that you needed, but this time with the header that you wanted.

This solution not only fixes the issue in Chromium based browsers, but also doesn't change the way Firefox, Safari and other browsers view your app.

✌️