Open in app ↗ Sign up Sign In

Search Medium

Published in The Startup

You have **2** free member-only stories left this month.
Sign up for Medium and get an extra one

Martina Ivaničová Follow
Aug 4, 2020 · 7 min read · · Listen

Save

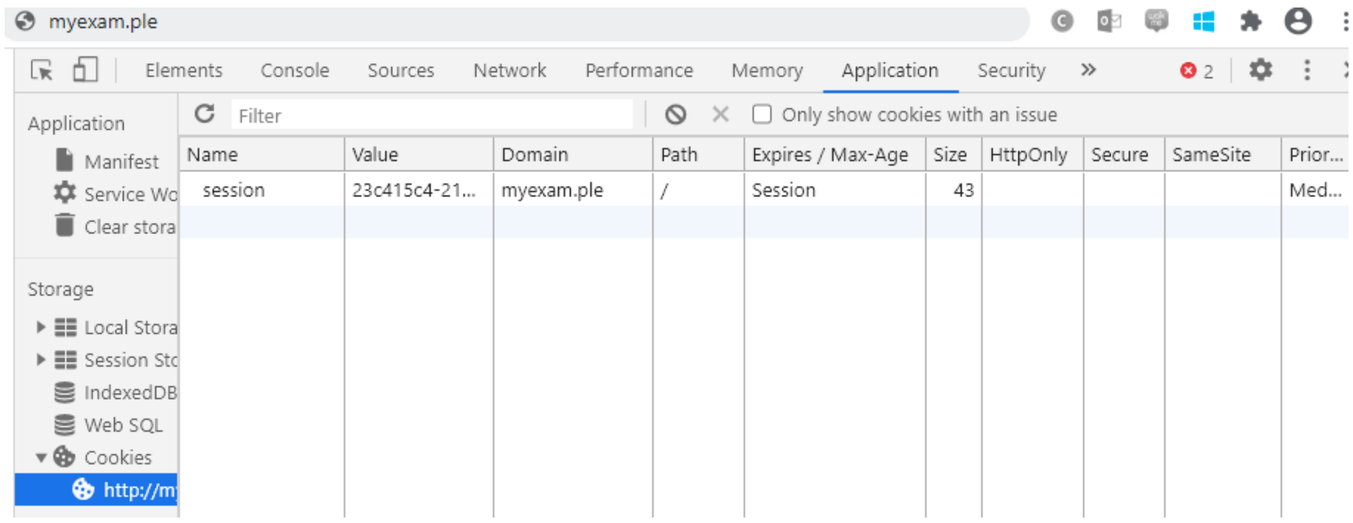
7 Keys to the Mystery of a Missing Cookie

Photo by Emiliano Vittoriosi on Unsplash

One day my colleague reported he couldn't access a certain website. On every attempt he had been redirected back to the login page. I browsed to that website and surprisingly everything was working fine for me.

We checked our browsers and found out that we both are using the same version of the Chrome. What went wrong was that in his case session cookie was not passed along the request to the backend service and the backend replied with http status code 401 — Unauthorized.

There are a couple of reasons why the browser would not attach a cookie to the request even if we are expecting it to do so. This incident and it's troubleshooting inspired me to sum them up here. What caused my colleague's issue is marked as 1. in this blog. Before we proceed to it let's start by hitting F12 and opening the developer console to see what cookies and their properties we have set.



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Prior...
session	23c415c4-21...	myexam.ple	/	Session	43				Med...

`Path` : if / the cookies will be sent for all paths

`HttpOnly` : if true, the cookie cannot be accessed from within the client-side javascript code.

`Secure` : cookie has to be sent over HTTPS

`SameSite` : `Lax` , `Strict` , `None` or not set. Instructs browser whether or not to sent cookie in case of cross-site requests

`Domain` : The domain for which the cookie is set and can be sent to.

`Max-Age` : Time to live of the cookie

1. SameSite attribute Defaults to Lax

SameSite is an attribute of a cookie which tells the browser whether to attach a cookie to the cross-site request.

There was a breaking change recently with Chrome and other browsers will probably follow this behaviour soon. The thing is that cookies which do not have the `SameSite` value explicitly set, were previously treated as `SameSite=None` .

Now, as of the version 80 of Chrome (canary released to gradually increasing population — that's why we were with my colleague experiencing different behaviour even if having the same version installed), they are treated as `Lax` so they are not sent along when e.g. XHR request is targeting the domain which is different then the origin.

If we are browsing the website http://www.example.com and the website triggers XHR request to http://myexam.ple/, cookies without `SameSite` which are stored in the browser for http://myexam.ple domain will not be sent along the request.

As of July 2020 Chrome started release of yet another closely related feature. If the cookie's attribute `SameSite` is `None` the cookie has to be set with flag `Secure` . If the cookie doesn't have the `Secure` flag, the browser ignores the `Set-cookie` server's response header and the cookie is not stored to the browser. If you got this wrong, you probably see in the the developer console following error message:

> *A cookie associated with a cross-site resource at https://myexam.ple/ was set without the `SameSite` attribute. A future release of Chrome will only deliver cookies with cross-site requests if they are set with `SameSite=None` and `Secure`*

More about `SameSite` attribute and cross-site requests could be found in the nice and explanatory blog post here.

Troubleshooting tip: In Chrome type in the URL chrome://flags and disable these two flags: `SameSite by default cookies` and `Cookies without SameSite must be secure`. If this helped, you now know the issue and you can apply the fix.

Solution tip: Modify the server code to explicitly set the cookie's `SameSite` attribute to `None`.

This is a sample code of the controller written in Java Spring Boot of how to add a server response header to set a cookie named "myCookie" of value "hello" with the attribute `SameSite=None` and flag `Secure`.

```
        @CrossOrigin(origins = {"http://example.com" })
        @RequestMapping(method = RequestMethod.GET, value = "/api")
        @ResponseBody
        public Hello HelloController(HttpServletResponse res) {
                res.setHeader("Set-Cookie", "mycookie=hello"; Secure; HttpOnly; SameSite
                res.setStatus(200);
                return new Hello("response body of my request");
        }
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

Note: `SameSite=None` opens the door to the cross-site request forgery vulnerability. It's strongly suggested to consider having some other CSRF protection in place.

2. withCredentials is not Set to True

When the website http://example.com which we are browsing triggers a `POST` XHR request to http://myexam.ple/api, the browser identifies this as cross-site request and it will not attach cookies and authorization headers to the request unless the default behaviour is overridden by setting the `withCredentials` property of XHR request to `true`.

Solution tip: Modify your client code, so the XHR request has an option `withCredentials` set to true. Here is an example of how to set the `withCredentials` property in a client app written in Angular.

```
async getData(): Promise<ApiResponse> {
  let defaultHeaders = new HttpHeaders();
  defaultHeaders = defaultHeaders.append('Accept', 'application/json');
  const requestOptions = { headers: defaultHeaders, withCredentials: true};
  const apiResponse: ApiResponse = await this.http.get<ApiResponse>(`//myexam.ple/api`
  return apiResponse;
}
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

There are a couple of things you have to make sure in order tomake `withCredentials :true` take effect. They are listed in the next section.

3. Preflight Request Blocks Credentials

Let's have a closer look to what happens if we are browsing https://example.com and the website makes a `POST` request to http://myexam.ple/api.

The browser detects the cross-site requests and before proceeding with the `POST` to myexam.ple/api the browser automatically fires the `OPTIONS` preflight request.

Note, that preflight requests are not triggered by `GET` requests since `GET` requests by definition are not intended to modify user data. Preflight requests are triggered by `POST`, `PUT` and `DELETE` requests.

`OPTIONS` response headers instructs a browser how to compose the actual `POST` request. For `withCredentials:true` to take the effect (discussed in #2), `OPTIONS` response headers must not be `ACCESS-CONTROL-ALLOW-ORIGIN:*`, instead it has to explicitly list origins e.g. `ACCESS-CONTROL-ALLOW-ORIGIN:http://myexam.ple`

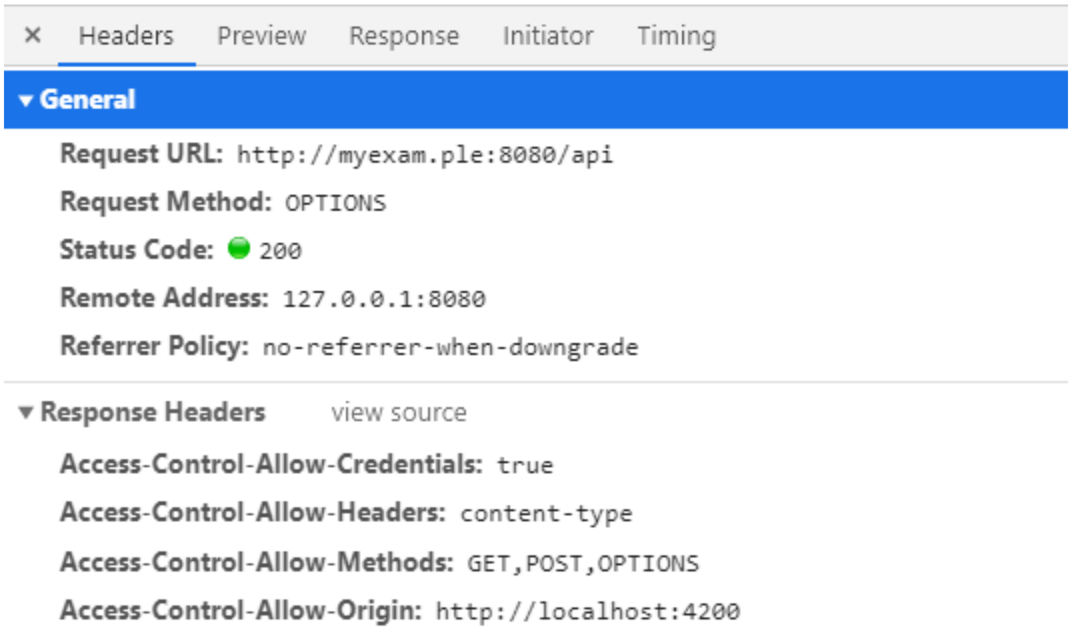
If you got this wrong you probably see in the developer console this error message:

> *The value of the 'Access-Control-Allow-Origin' header in the response must not be the wildcard '*' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute.*

It makes sense, you don't want an arbitrary webpage called your API with credentials, don't you?

Check out the `OPTIONS` response header `ACCESS-CONTROL-ALLOW-CREDENTIAL` whether it is set to `true`. If the server doesn't allow credentials being sent along, the browser will just not attach cookies and authorization headers. So this could be another reason why the cookies are missing in the `POST` cross-site request.

Troubleshooting tip: open the developer console and check in the Network tab what are the response headers from `OPTIONS`.



Request URL: http://myexam.ple:8080/api
Request Method: OPTIONS
Status Code: ● 200
Remote Address: 127.0.0.1:8080
Referrer Policy: no-referrer-when-downgrade

Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Methods: GET,POST,OPTIONS
Access-Control-Allow-Origin: http://localhost:4200

Solution tip: On your server code, set the appropriate response headers. Here is the example of how this could be done in Java Spring Boot.

```
        @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList("http://example.com")); //only requests
        config.setAllowedMethods(Arrays.asList("GET","POST"));
        config.setAllowedHeaders(Arrays.asList("content-type"));
        config.setAllowCredentials(true);   //this is important in preflight request (O
        UrlBasedCorsConfigurationSource configSource = new UrlBasedCorsConfigurationSou
        configSource.registerCorsConfiguration("/**", config);
        return configSource;
    }
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

4. Path is not Matching

If the cookie was set for `Path /` it means that it is sent along all the requests targeting the domain for which it was set, e.g myexam.ple/customers. However if the cookie `Path` was set to /api, the cookie will be sent only when request to path starting myexam.ple/api is made.

Troubleshooting tip: open the developer console, navigate to Application>Cookies and edit the path attribute directly in there to see if this helps

Solution tip : Fix the code to set the cookies with matching `Path` .

5. Domain is not Matching

The key aspect of the browser security is that a cookie is only sent over to the host for which it was set.

When setting a cookie for myexam.ple with `Domain` attribute omitted, it means that all requests to myexam.ple will be with the cookie, however requests to the subdomain subdomain.myexam.ple will not have the cookie attached.

If the `Domain` is specified e.g. `Domain=myexam.ple` , it means that a cookie will be sent with the request to myexam.ple as well with the request to subdomain.myexam.ple.

Troubleshooting tip: open the developer console, navigate to Application>Cookies and edit the `Domain` attribute directly in there to see if this helps.

Solution tip: Change the code where you are setting the cookie to set the `Domain` attribute accordingly.

6. Cookie Name is Prefixed with '__Host'

Cookie prefixes is an additional way to instruct a browser how the cookie should be treated.

Cookies prefixed with `__Host` are sent only to the host which set the cookie and never sent to subdomains. So if the cookie `__Host_mycookie` is set for http://example.com and your request targets http://sub.example.com/api the cookie is not attached.

Cookie prefixes are well explained in this post.

Note, that `__Host` cookies cannot have `Domain` attribute set.

7. Cookie is Expired (Expires/MaxAge is in Past)

Last point, that a cookie could be expired, is so obvious that I even hesitated to mention it here. Everything has to come to its end, including the cookie's life and this blog post as well…

Thank you for reading and if you faced any other reason for cookie being lost I would love to hear back from you.

Resources

- https://www.chromestatus.com/feature/5088147346030592 — Chrome's release notes on SameSite behaviour
- https://web.dev/samesite-cookies-explained/ — SameSite attribute explained
- https://tools.ietf.org/html/draft-ietf-httpbis-cookie-prefixes-00 — Cookie prefixes and reasons for their introducing explained
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies — All about

Programming Cookies Chrome Angular Security

818 12

Sign up for Top 5 Stories

By The Startup

Get smarter at building your thing. Join 176,621+ others who receive The Startup's top 5 stories, tools, ideas, books — delivered straight into your inbox, once a week. Take a look.

Your email

Get this newsletter