Stackoverflow

Question : 78761021 Session id undefined express

Extended comments : 25-Jul-24

Reason : The objective of the following document is to give extended comments on the answer. It is expected that it may help both of us to convey the matter in a better and extended way.

Extended comments:

We shall see below some live cases which may help us to firm up our understanding about a session-cookie rejection

or a session-cookie life-cycle in general. These test cases have been conducted in a production server hosted with render.com.

Before we move on to the cases, let us reiterate a session-cookie life-cycle in a nutshell.

1. A session-cookie is generated at the server side.
2. It is generated at the server side on a fresh or new request from a client.
3. A server identifies a fresh or a new request by looking at the session-cookie present in a request.
4. If there is no session-cookie present in a request, it is treated as a fresh request, and a new session object is
5. created for it.
6. If a request has session-cookie present in it, then the same request will be treated as a subsequent request

from the same client.

1. The server then sends back the session-cookie to clients through its responses.
2. The client will store the session-cookie permanently till the end of a session.
3. The client will include the session-cookie on each and every request to the same server or the same URL.
4. The client will identify the respective session-cookie by matching the Request URL and the domain specified in a session-cookie. This is where the application of the domain attribute in a session-cookie comes into play.

It means the domain attribute of a session-cookie set by the server at the creation of a session-cookie,

will be referenced by a Browser on each and every time it makes requests with the respective URL.

In short, this domain attribute acts as a tag to identify the respective session-cookie. A Browser may have a number of session-cookies stored in it from various applications it may run.

1. A session-cookie with no lifetime will be removed by a Browser at the time of ending the browser session.

Basic information about the cases.

1. All test cases have been conducted using the same request shown below.

Request URL : <https://www.test1wedothebest.lat/cookie>

1. All test cases have been conducted directly from a Browser. It means there is no custom client application used to access the server api. This is to keep the testing environment simple and straight with no CORS complexities.
2. The server has only a single api referenced as cookie.
3. A full listing of the server code has been listed at the end.

Case a : no domain specified, coded as below:

app.use(

session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

cookie: { secure: true },

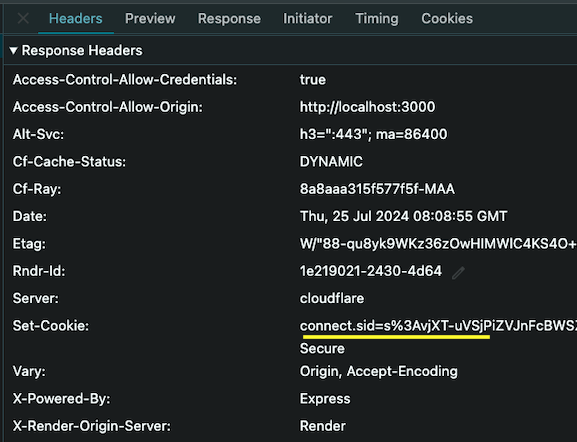
})

);

Results:

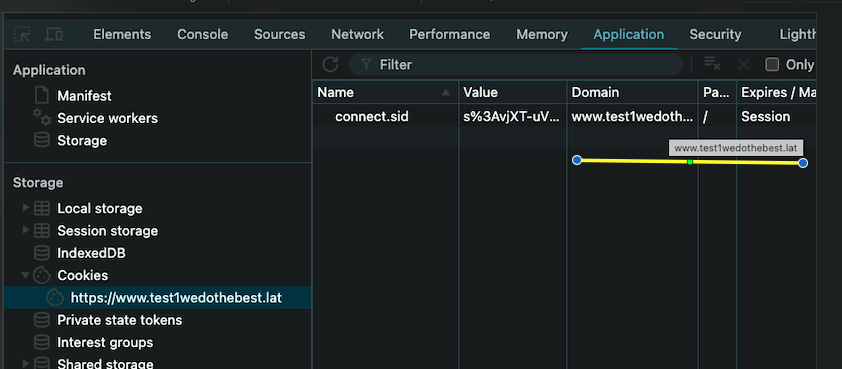
1) The server could send the cookie to the Browser. Please see below the response received at the Browser from the server,

it includes the session-cookie.

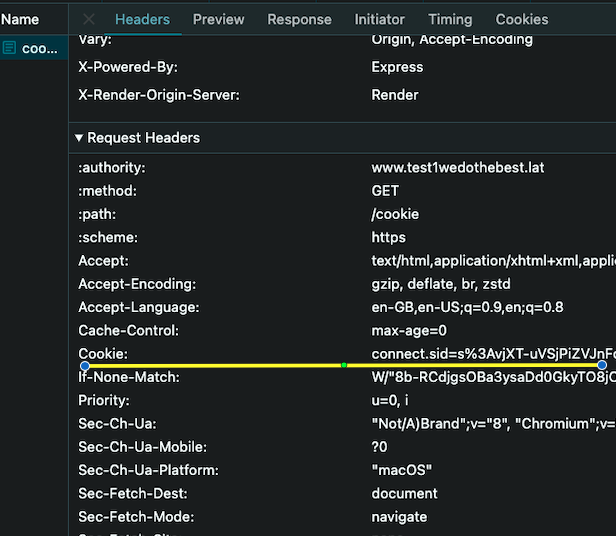


2) Please pay close attention here, there is no domain visible in the cookie displayed above. The server did not have that information and therefore it did not set it. Now the Browser will do the rest of work. As there is no domain set in the cookie it received, the Browser cannot leave this field blank as this piece of text is vital to identify the cookies and include them in the next request it may ever make with the same Request URL. Therefore the Browser will read the domain of the request URL which is in this case - test1wedothebest.lat, and set it to the cookie as its domain. By this it is assured that the Browser can later relate the same cookie with the same URL.

3) As a result of step 2, the cookie gets stored in Browser with the proper identification as shown below. Please take note of the domain set by the Browser.



4) The browser could include the same cookie in its subsequent request to the same URL as below. By this way, one cycle of session-cookie has been successfully completed and the same will repeat for each and every request-response cycle.



Observation: When there is no domain specified in the session cookie at the server side at its creation, then the default - the request’s host name will be assigned as the cookie's domain name. Therefore in this case the request’s host name - test1.wedothebest.lat,

is set as the cookie’s domain name. Since the cookie has been set with this particular domain name,

all subsequent requests to the same domain will include this cookie in its request. This is what has been happening.

Case b : a domain which does not exist specified, coded as below:

app.use(

session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

domain : ‘somenonexistingdomain.com’

cookie: { secure: true },

})

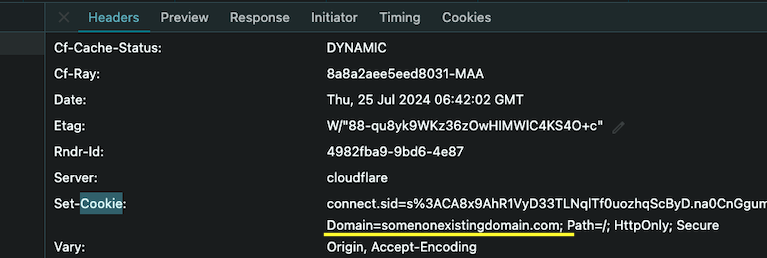
);

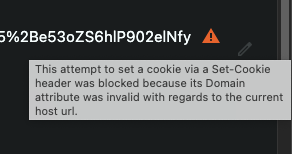
Results:

1) Server could send a cookie to the Browser.

2) And the Browser has received it. Please see the response received

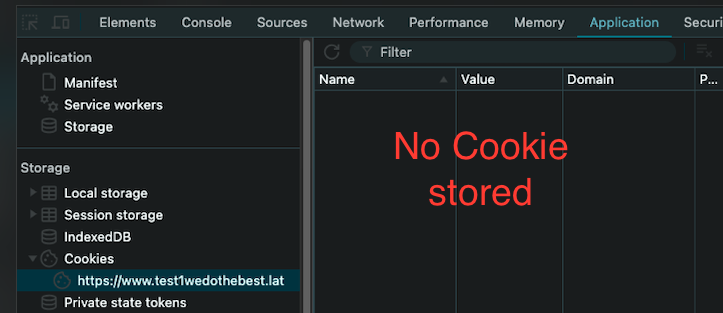
in the Browser where the same cookie has been shown. However there is a warning symbol as well. A warning symbol displayed on top of the cookie text. While hovering over the warning, the below popup will be shown.





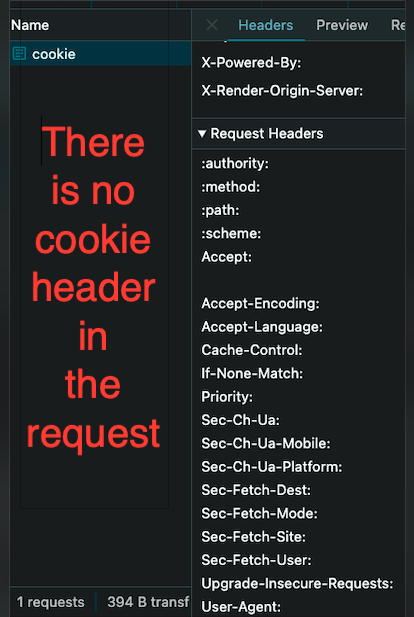
3) As a result of the blocking, the same cookie has not been stored in the Browser in the normal way.

Please see the cookie storage.



4) Therefore in the subsequent request to the same URL, the browser could not include the same cookie. Please see the

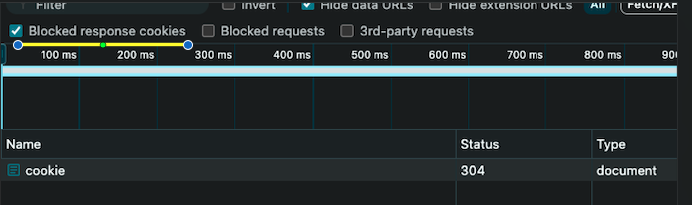
request details of the next request, the cookie is not present in it.



Observation: When there is a domain specified in the session cookie, then the server simply takes it for granted and

sets the same as the domain of the cookie. However, when the same cookie is reached at the Browser, the Browser does the detection. The Browser checks the cookie’s domain with the request URL. If it does not match, then it will block. Therefore this process is a simple string comparison, and it does not involve or require any additional validations like whether it is an active domain or not.

Note: Such exceptions are shown in Chrome separately. We can see the blocked response in Chrome via the option - Blocked response cookies.



case c : a third party domain specified, coded as below:

app.use(

session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

domain : ‘google.com’

cookie: { secure: true },

})

);

Results and Observations: Same as that of Case b.

Code listing:

const express = require('express');

const cookieParser = require('cookie-parser');

const session = require('express-session');

const app = express();

app.set('trust proxy', 1);

// case a

// app.use(

// session({

// secret: 'keyboard cat',

// resave: false,

// saveUninitialized: true,

// cookie: { secure: true },

// })

// );

// case b

// app.use(

// session({

// secret: 'keyboard cat',

// resave: false,

// saveUninitialized: true,

// domain : ‘somenonexistingdomain.com’

// cookie: { secure: true },

// })

//);

// case c

app.use(

session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

domain : ‘google.com’

cookie: { secure: true },

})

);

app.use(cookieParser());

app.get('/cookie', (req, res, next) => {

let response;

const addinfo = `protocol : ${req.protocol}, host : ${

req.hostname

}, origin : ${req.get('origin')}`;

if (req.session?.somecookie) {

response = `Same cookie: A cookie received and the same sent to client - ${addinfo}`;

} else {

req.session.somecookie = 'cookie text';

response = `New cookie: A new cookie created and sent to the client - ${addinfo}`;

}

res.send(`<h1>${response}</h1>`);

});

app.listen(4000, () => {

console.log(`L@4000`);

});