GraphQL — Common vulnerabilities & how to exploit them

+Bilal Rizwan

Follow

Apr 4 · 10 min read

Hello there! how you doin? , Bilal Rizwan here & I hope everyone is safe in this time of crisis and making complete use of your quarantined time to learn new things and expand your skill.

What is this post about ?
Many of you might have now seen GraphQL being used in a lot of web applications, some of you might have recognized right away that its graphql and probably tried searching for what you can do with it some might not have realized that the request is something called GraphQL request.
In this post I'll try to highlight the common misconfigurations in the usage of GraphQL and how they can be exploited.

For those who don't know what GraphQL is its request looks like this.

GraphQL sample request

It has some curly brackets and \n characters. If you see something like that then most likely its GraphQL.
Lets first start off by understanding what GraphQL actually is knowing this will help up better form exploits.

What is GraphQL ?

Well Simply put GraphQL is an alternative API standard like REST and SOAP. It is basically a Query language for APIs used to interact with the APIs and to fetch data from the backend through APIs. It can do everything REST API can but in a much more efficient and controlled way. GraphQL solves a lot of the problems faced while using REST like fetching more data than it should or the need to have a new endpoint for every call.

The following example should clear the difference between GraphQL and REST API.

In REST API we would typically be using `/users/<id>`endpoint to fetch user data. Secondly, there's likely to be a `/users/<id>/posts` endpoint that returns all the posts for a user. The third endpoint will then be the `/users/<id>/followers` that returns a list of followers per user

REST API functionality

In GraphQL however there is only one endpoint where we send a query which includes concrete data requirements the server then responds with the data requirements.

Suppose we want to fetch the user id's from the system we can make a query for it like this

Fetch id

Now what if we also want user email's? well unlike REST API we can just specify that in a new line on the same endpoint in the request is being sent to the server and its just that simple

What if we also want the names ? I am sure you are able to now able to figure it out.

And that is the beautify of Graphql you can just specify what you want in a granular fashion.
Now that you know what GraphQL is all about lets move on.

Common Misconfigurations in GraphQL

Thing to understand here is that GraphQL like any other REST API is vulnerable to many attacks the same attacks the REST API might be prone to. I'll list some of them below but the most interesting thing and the reason of making this entire post is the infamous **Introspection query** bug.

Introspection query: *Simply put is a way to query the server for its GraphQL back-end schema and to get a complete documentation and list of what API calls are available in the back-end. This is originally meant to be used internally.*

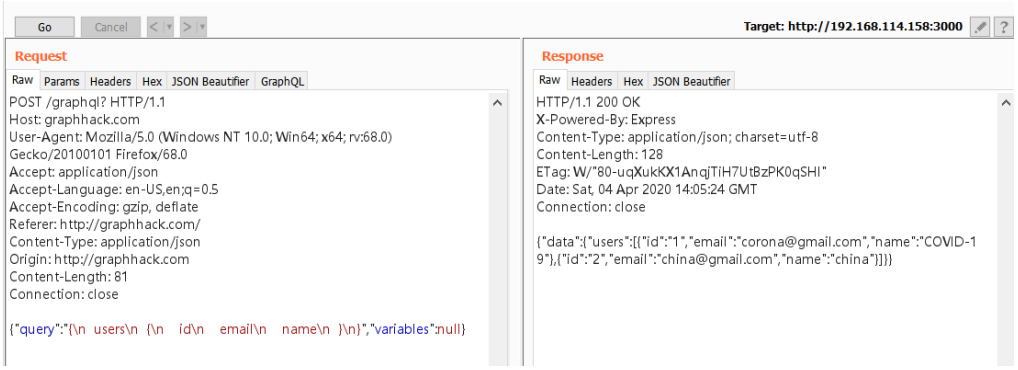
The introspection query should only be allowed internally and should not be allowed to the general public. If we can fetch the entire back-end API documentation and calls available on a server then that can be very

dangerous is many cases what if we could get our hands on some API calls only meant to be used internally maybe we find a call to enable debugging or perhaps an API calls to delete users there is so much damage that can be done if entire back-end can be fetched.

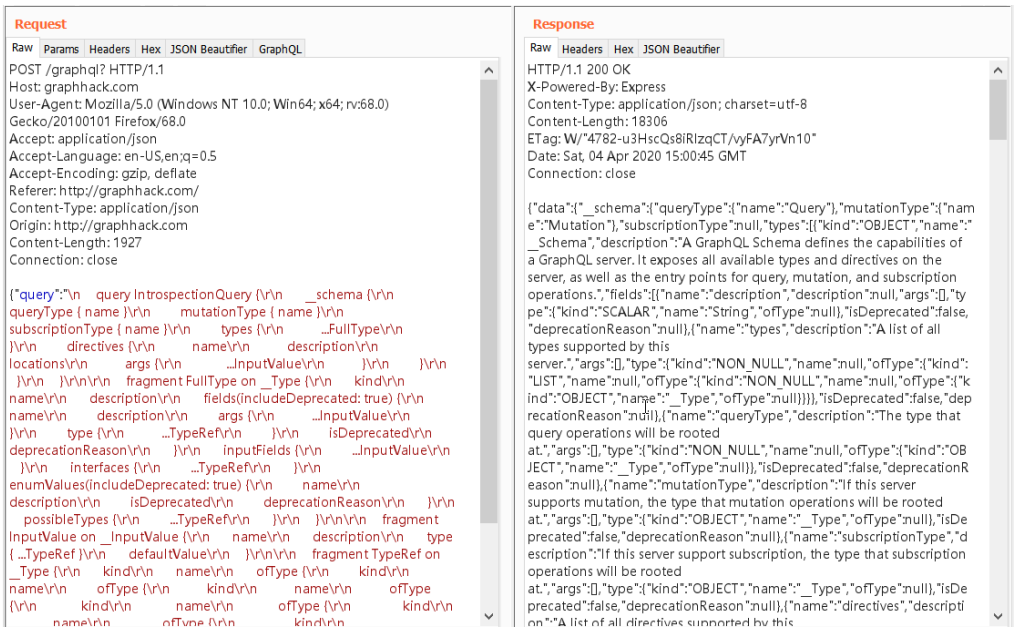
Lets see how this is done.

To test a server for GraphQL introspection misconfiguration:
1) Intercept the HTTP request being sent to the server
2) Replace its post content / query with a generic introspection query to fetch the entire backend schema
3) Visualize the schema to gather juicy API calls.
4) Craft any potential GraphQL call you might find interesting and HACK away!

Suppose your target web app is making an GraphQL call you can simply change its query with a GraphQL Introspection query as follows.



Original API



Original API call

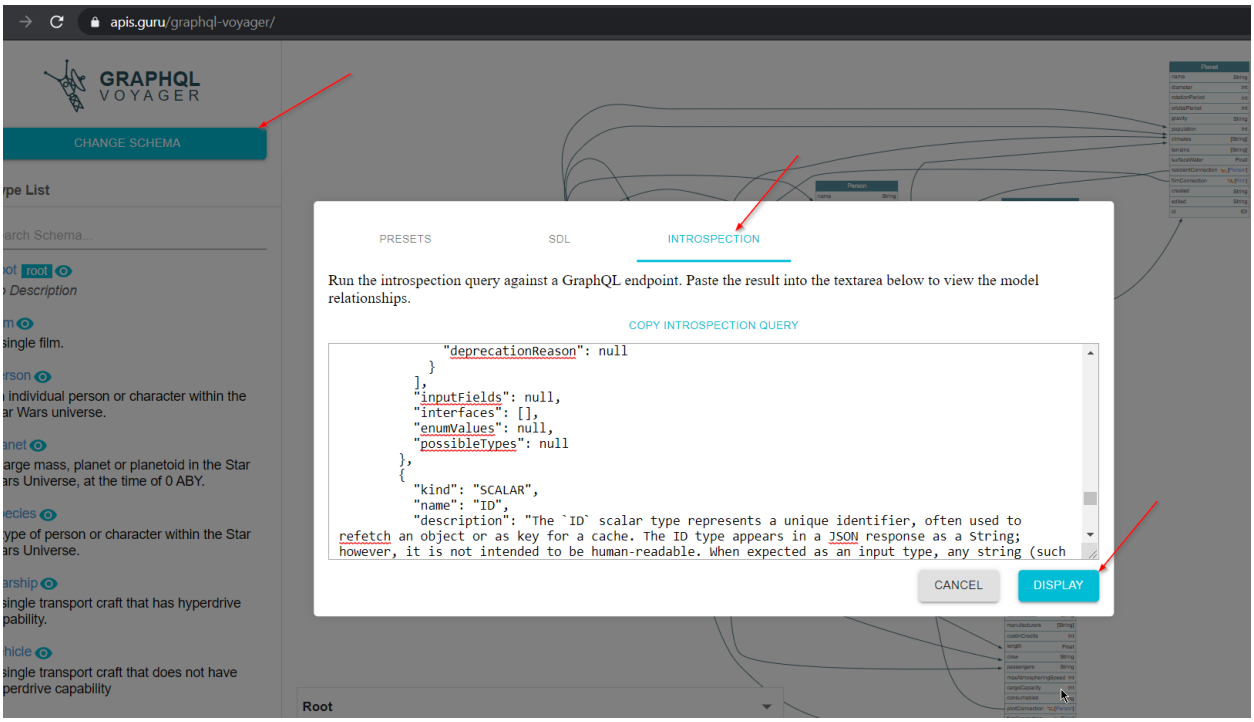
GraphQL Introspection Query Leaking back-end schema

Just replace the POST contents with the following query:

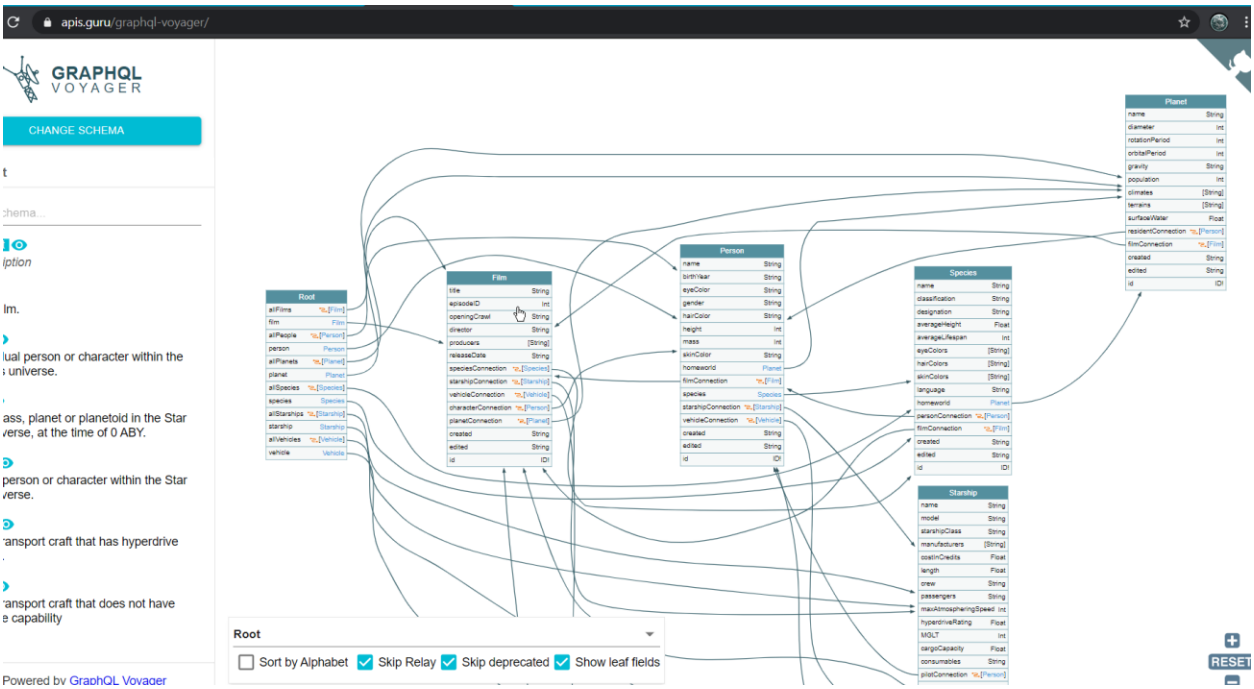
```
{"query":"\n    query IntrospectionQuery {\r\n      __schema {\r\n
queryType { name }\r\n        mutationType { name }\r\n
subscriptionType { name }\r\n        types {\r\n
...FullType\r\n        }\r\n        directives {\r\n          name\r\n
description\r\n          locations\r\n          args {\r\n
...InputValue\r\n          }\r\n        }\r\n      }\r\n    }\r\n\r\n
fragment FullType on __Type {\r\n      kind\r\n      name\r\n
description\r\n      fields(includeDeprecated: true) {\r\n
name\r\n        description\r\n        args {\r\n
...InputValue\r\n        }\r\n        type {\r\n
...TypeRef\r\n        }\r\n        isDeprecated\r\n
deprecationReason\r\n      }\r\n      inputFields {\r\n
...InputValue\r\n      }\r\n      interfaces {\r\n
...TypeRef\r\n      }\r\n      enumValues(includeDeprecated: true)
{\r\n        name\r\n        description\r\n        isDeprecated\r\n
deprecationReason\r\n      }\r\n      possibleTypes {\r\n
...TypeRef\r\n      }\r\n    }\r\n\r\n    fragment InputValue on
__InputValue {\r\n      name\r\n      description\r\n      type {
...TypeRef }\r\n      defaultValue\r\n    }\r\n\r\n    fragment
TypeRef on __Type {\r\n      kind\r\n      name\r\n      ofType {\r\n
kind\r\n        name\r\n        ofType {\r\n          kind\r\n
```

```
name\r\n          ofType {\r\n            kind\r\n            name\r\n
ofType {\r\n              kind\r\n              name\r\n
ofType {\r\n                kind\r\n              name\r\n
ofType {\r\n                  kind\r\n              name\r\n
ofType {\r\n                    kind\r\n              name\r\n
}\r\n                }\r\n              }\r\n            }\r\n
}\r\n        }\r\n      }\r\n    }\r\n  ","variables":null}
```

Now when you do this The response might be quite big and hard to comprehend. The best way to understand the schema is to visualize it. That can be done by copying the entire response body and using [this](https://apis.guru/graphql-voyager/) website https://apis.guru/graphql-voyager/ click on the Change schema button and go in the introspection tab then paste the introspection query there



after you paste the schema in there click Display and voila you'll be presented with a visualization of the entire back-end and API calls available



Backend schema of GraphQL

Now since we have the entire API calls list we can go over it and easily try to figure out if there are any sensitive API calls that can be abused. This is the most prevalent type of bug found in GraphQL back-ends which can lead to quite critical scenarios.

I kinda feel that up till this point you might not have completely synced in the true impact of this bug so lets take an example For those of you who already know how to look at a graphql schema and craft graphQL queries reading any further will not be quite useful but for people who have gotten the schema visualized it spotted something sensitive looking and want to exploit it or just want to understand the true impact of introspection then read on ahead.

Learning GraphQL Query formation:
GraphQL has 3 basic type of queries or components

Queries

Similar to GET request in REST API, queries are used to fetch data.

Mutation

Used to create, edit and delete data.

Subscriptions

Used for real time communication. We won't be focusing on this.

All of the graphQL queries are written in a json format with line breaks and curly brackets.

Lets take an example and use a Web application which has 2 Features

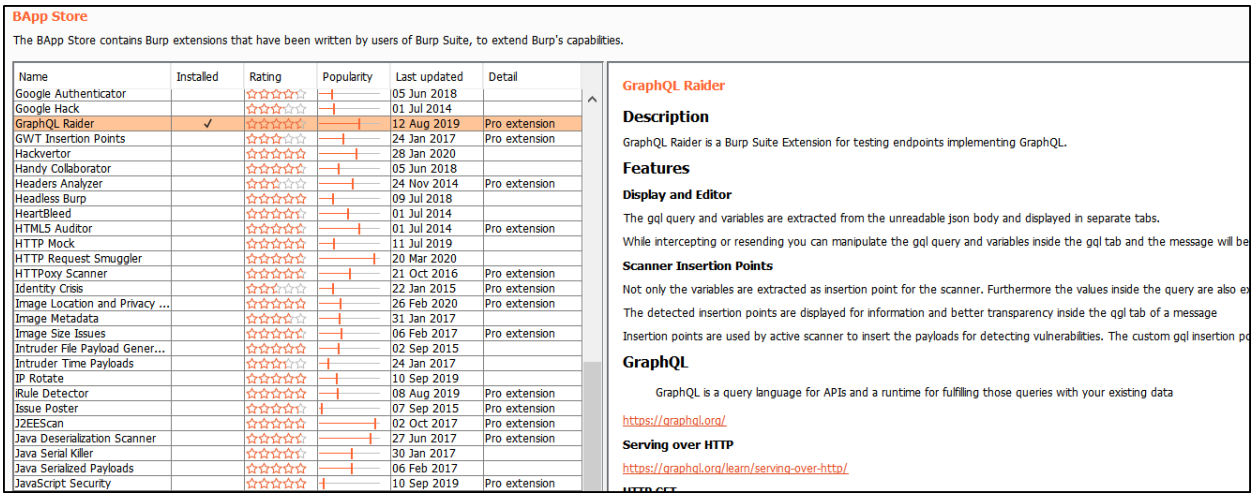
1) List users

2) Add Users

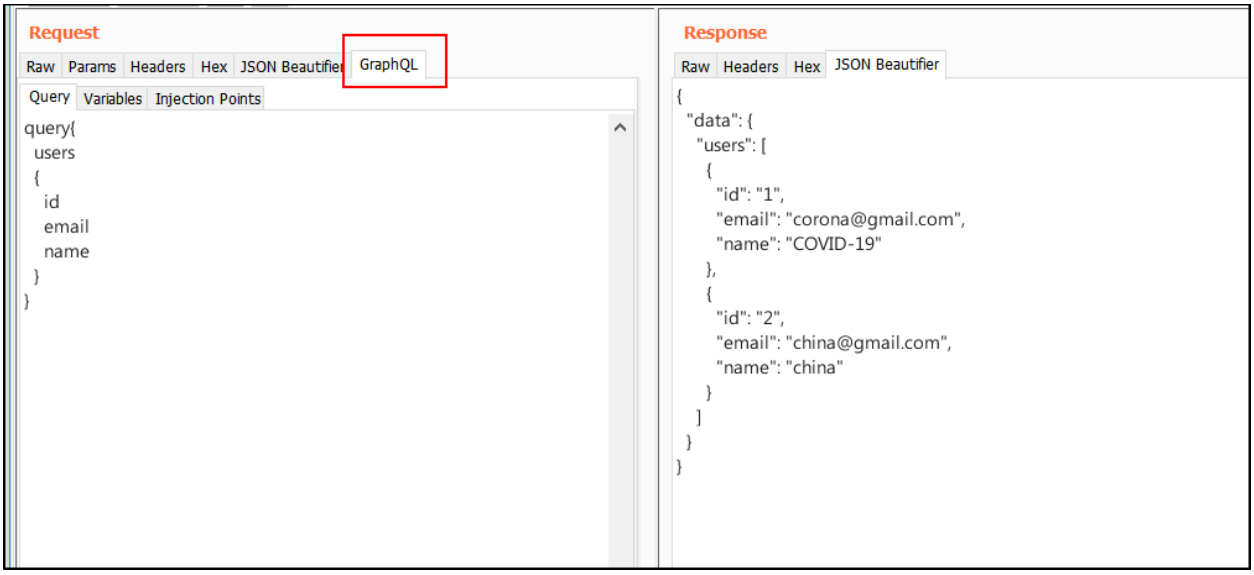
We will then use introspection query to find bugs in the application and this is something commonly found in web Apps.

GraphQL call to list users

The picture above is what you would normally be seeing now to make this better you can use a burp plugin like JSON Beautifier or better yet one more specific for GraphQL known as GraphQL Raider. After you install it you'll have a tab just like JSON beautifer where ever GraphQL



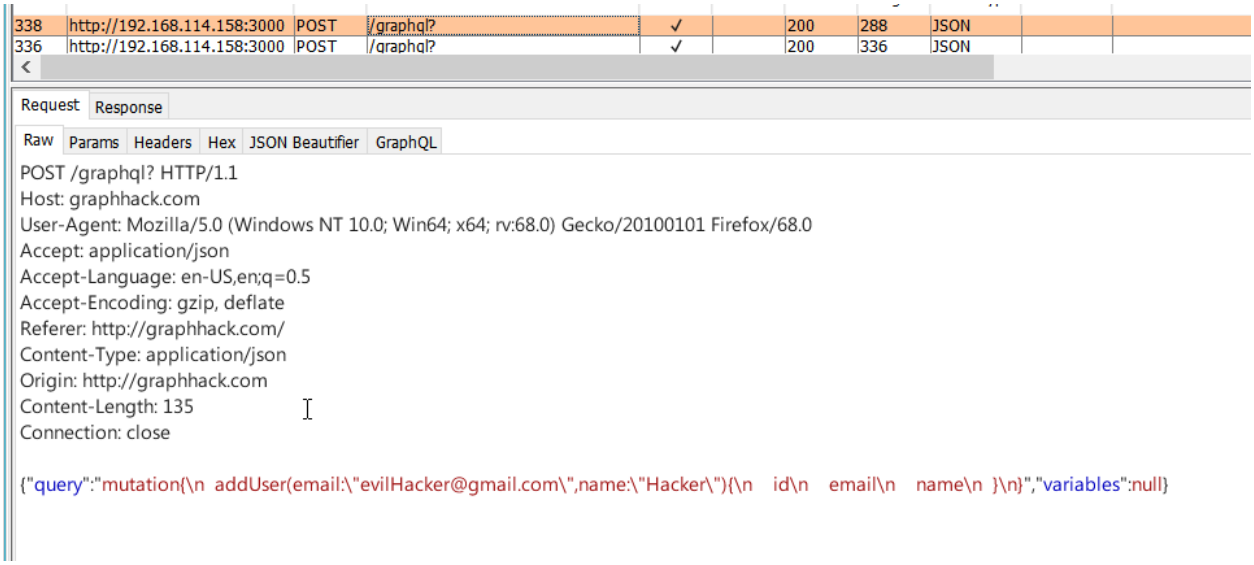
Now the above can be viewed in a much much better way



From the able images you should be able to easily understand the graphQL query structure

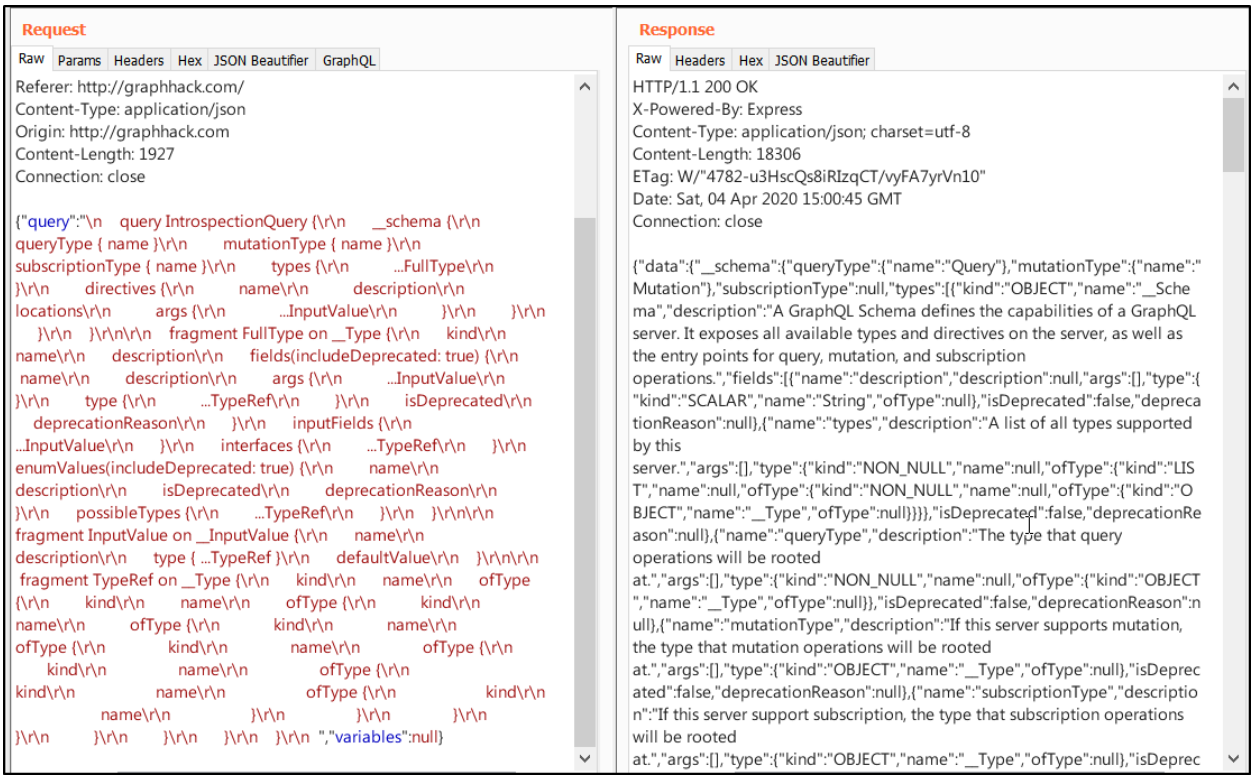
```
query{
  <FUNCTION NAME>
  {
     <COMPONENT>
     <COMPONENT>
  }
}
```

Simple as that. That's how Query works. Thats the first function the website provides
The App also lets us Add users lets try that out.

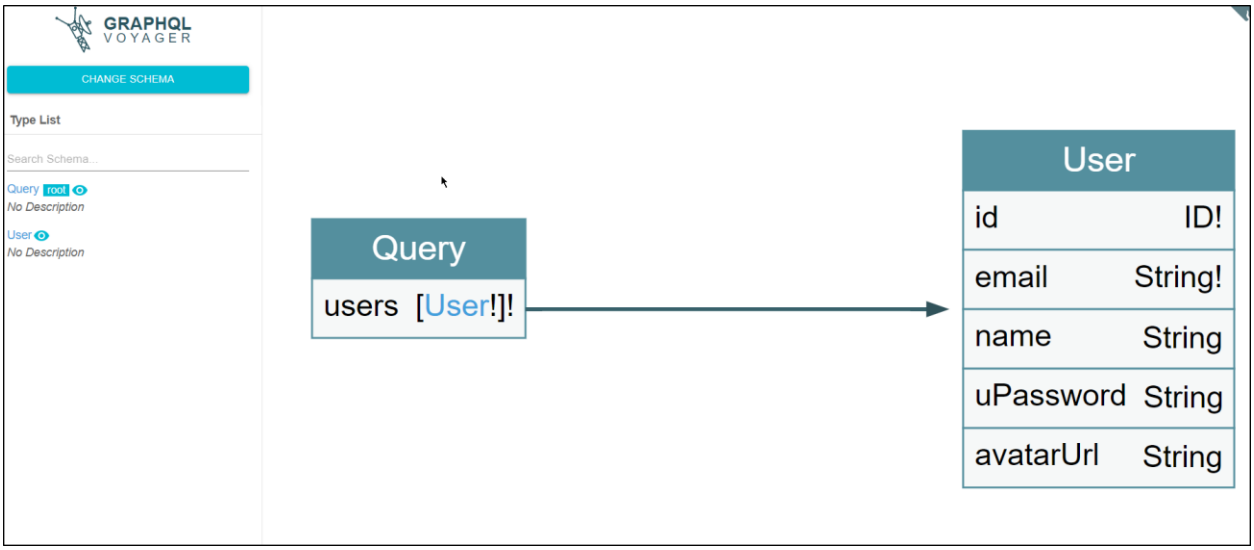


Now while we are able to view these calls through burp and generate them by clicking a button on the front end how do we find something more juicy ?
Well lets try introspection query.

Using Introspection query copy the response data and paste it in to voyager



Introspection query



Since this is a test App the schema is very small but look the image closely this is a very core and basic example how how you can find sensitive stuff using introspection as you can see in the picture the users function has

many fields some that we are already familiar with such as id, email, name but wait a min… what is this uPassword. Lets try to fetch it.

```
query{
  users
  {
    id
    email
    name
    uPassword
  }
}
```

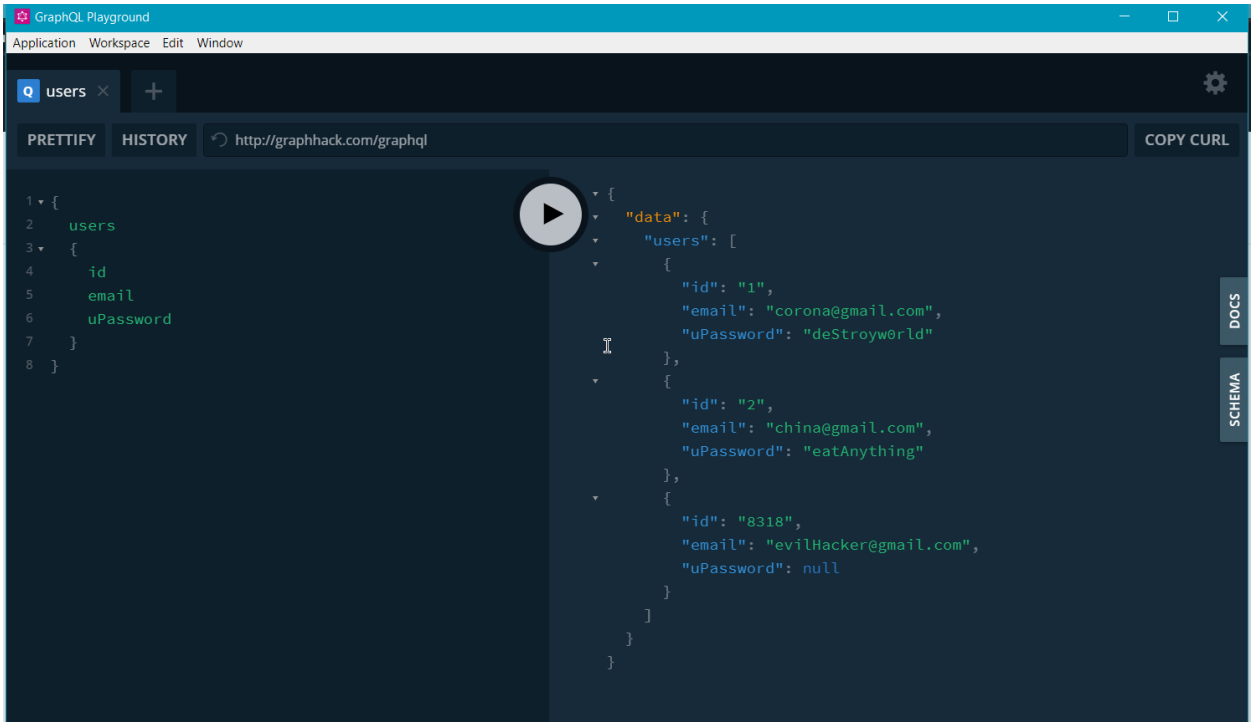
```
{
  "data": {
    "users": [
      {
        "id": "1",
        "email": "corona@gmail.com",
        "name": "COVID-19",
        "uPassword": "deStroyw0rld"
      },
      {
        "id": "2",
        "email": "china@gmail.com",
        "name": "china",
        "uPassword": "eatAnything"
      },
      {
        "id": "8318",
        "email": "evilHacker@gmail.com",
        "name": "Hacker",
        "uPassword": null
      }
    ]
  }
}
```

WOW!! we just fetched the passwords from the back-end. Why were we able to do this ? because in the backend there was a passwords field and we could use that in the graphql call this was only possible through the introspection.

— — — — — —

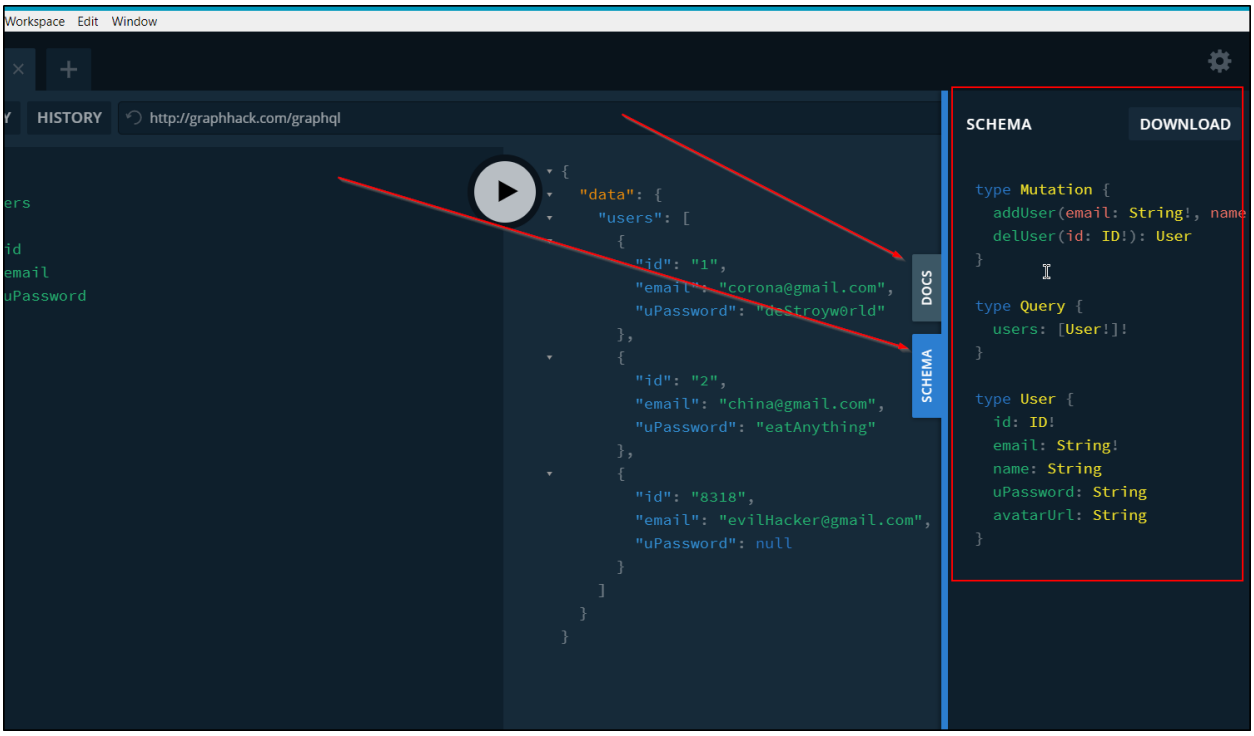
Now there is a small problem with the online graphQL visualization. It does not show mutations , did you spot that ? well then you have a keen eye. I for one did not spot that quickly it took a long time for me to know that and I am wondering how many bugs did I let pass . Sad nothing that can be done about that

The question now is how do we list the entire complete schema ? with the mutations ?Well I did some googling and found this
The graphQL playground download here

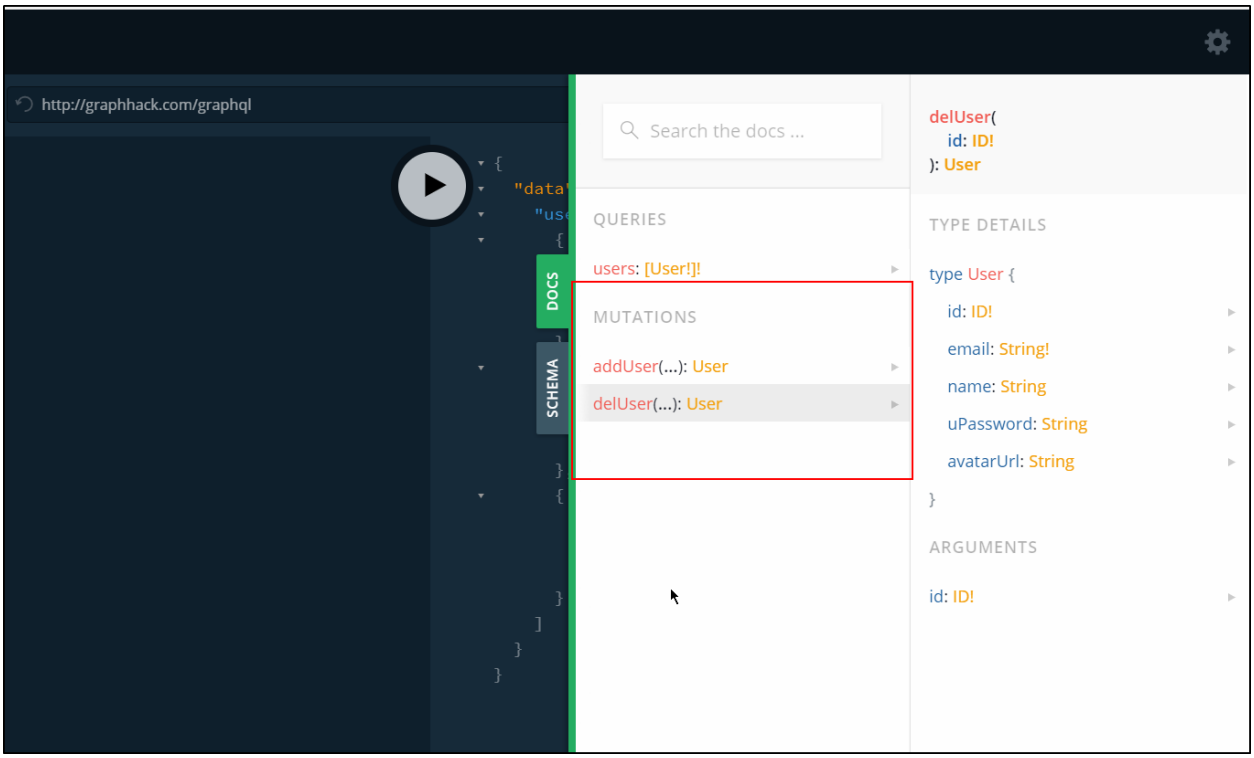


Now here is the great thing about the graphql Playground it actually utilizes the introspection query and makes a documentation out of it for you.

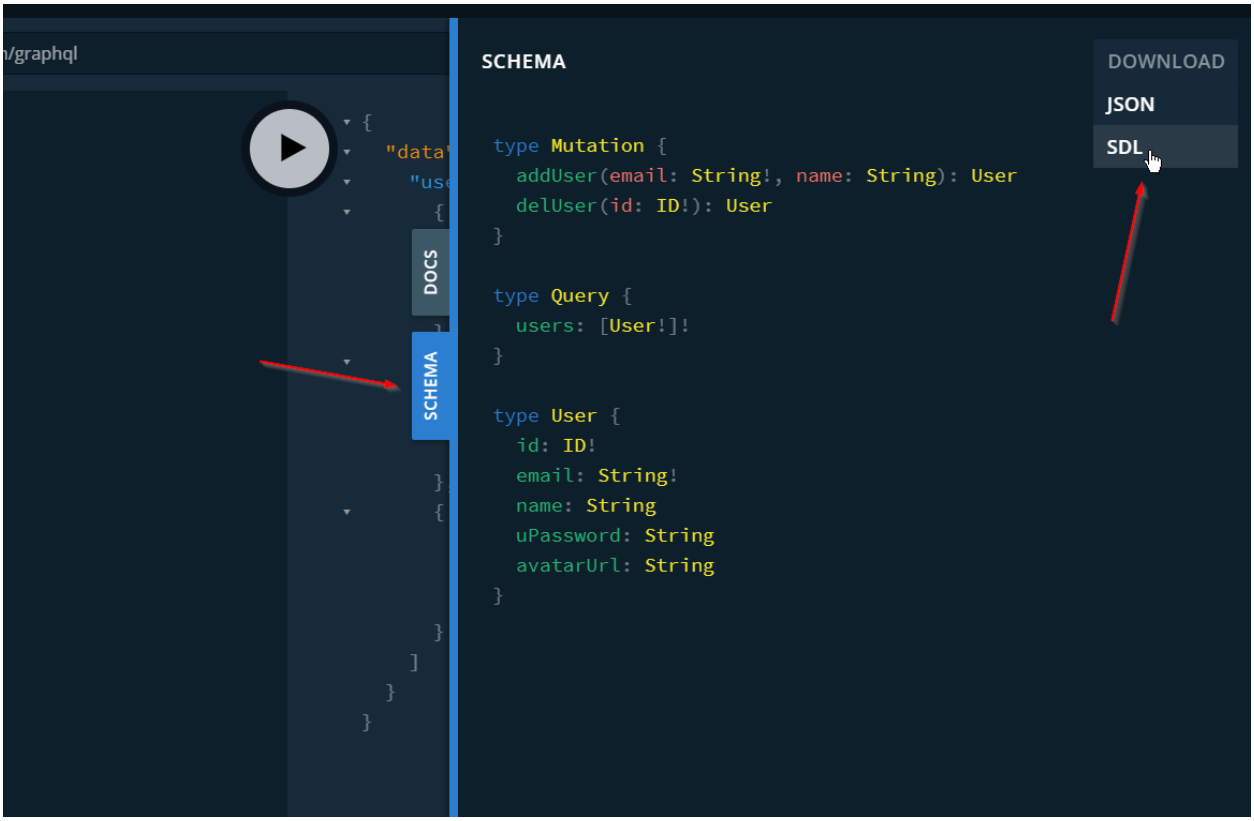
All you have to do is just download the Application paste in the URL of the graphql endpoint along with any cookies you have and voila!



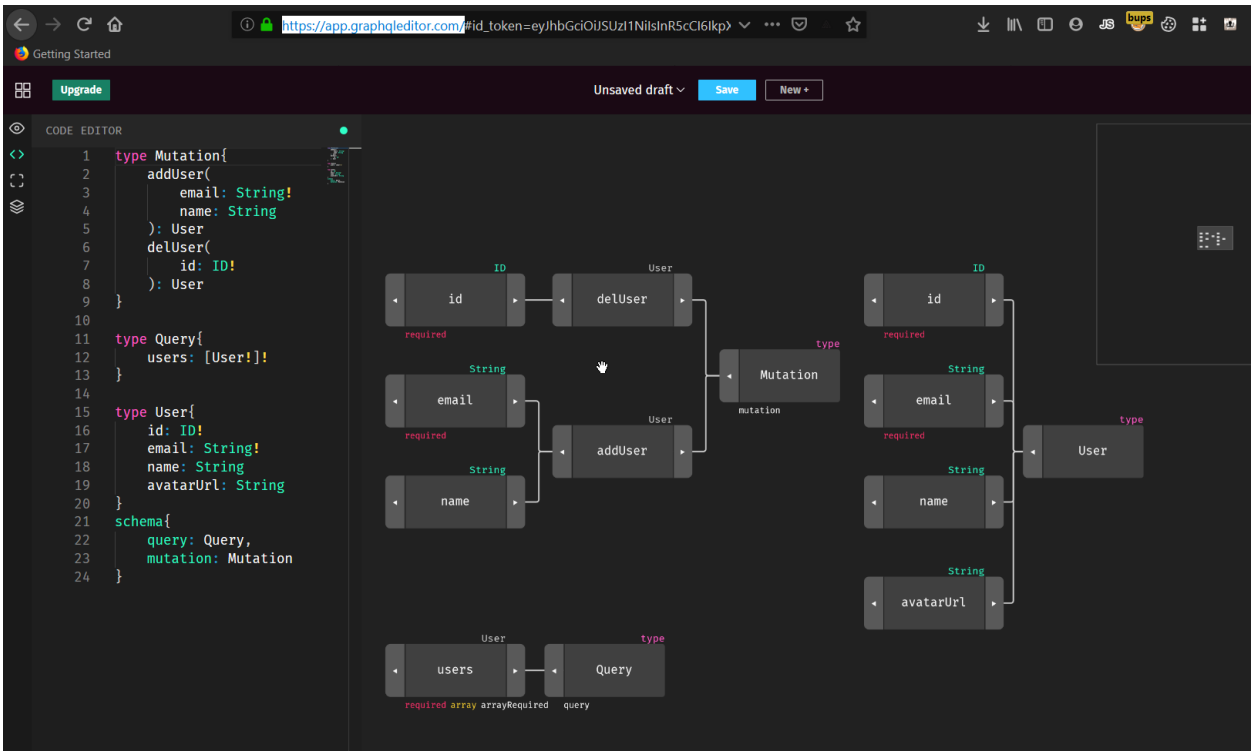
Nor normally you can just view all the schema here along with the mutations and its parameters



but I'd like to visualize this as well or atleast we should know how to so what you will do now is click on the Docs tab and download the schema in SDL format



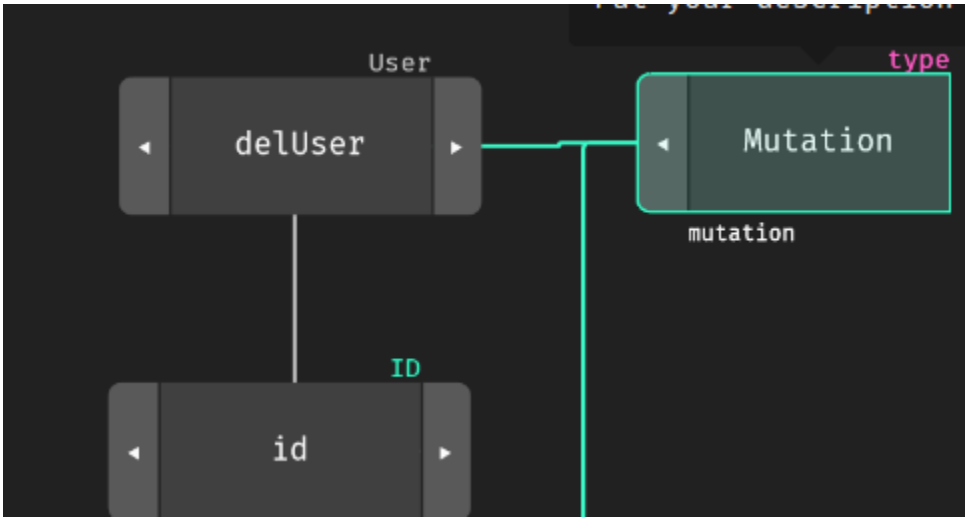
Once you have the schema head over to https://app.graphqleditor.com/ Sign in and you'll be given an option to paste the SDL file you downloaded after you do so the schema will be visualized for you along with Mutation calls.



Now would you look at that Beautiful isn't it ?

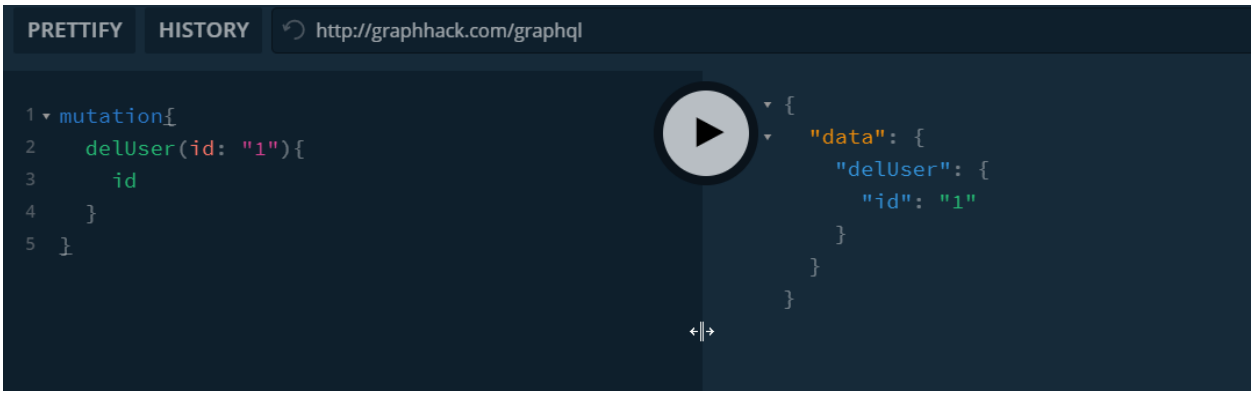
Alright viewing the mutation part we see another function not available on the front end the DeleteUser function and it even shows that it takes a single argument of ID lets try it out.
Lets delete user 1.

Passing the argument in like this by utilizing the visualization

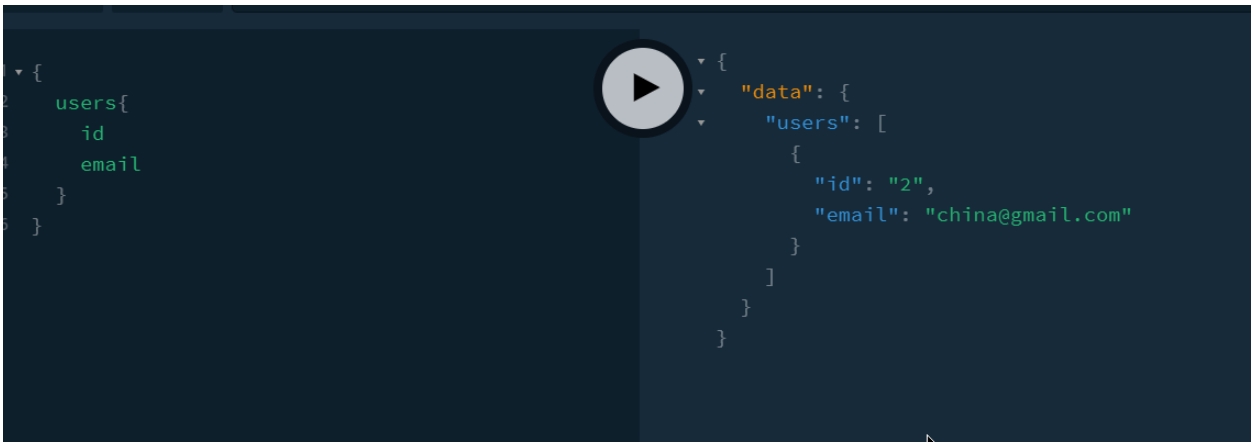



```
mutation{
  delUser(id: "1"){
    id
  }
}
```

Simple isn’t it ? If not then try to read the mutation part given above again and I am sure you will get it. After running the delUser query lets list the users again.



DelUser graphQL call



And as we can see the user 1 has been deleted.
This is just a simple example of the things that can be found using Introspection

Other bugs in Graphql:

Well just like any other REST API GraphQL is vulnerable to all the API bugs out there for example

1) IDORs

Where ever you see ID's you know what to do this bug is not related to GraphQL but is an Authorization bug

2) Bruteforce

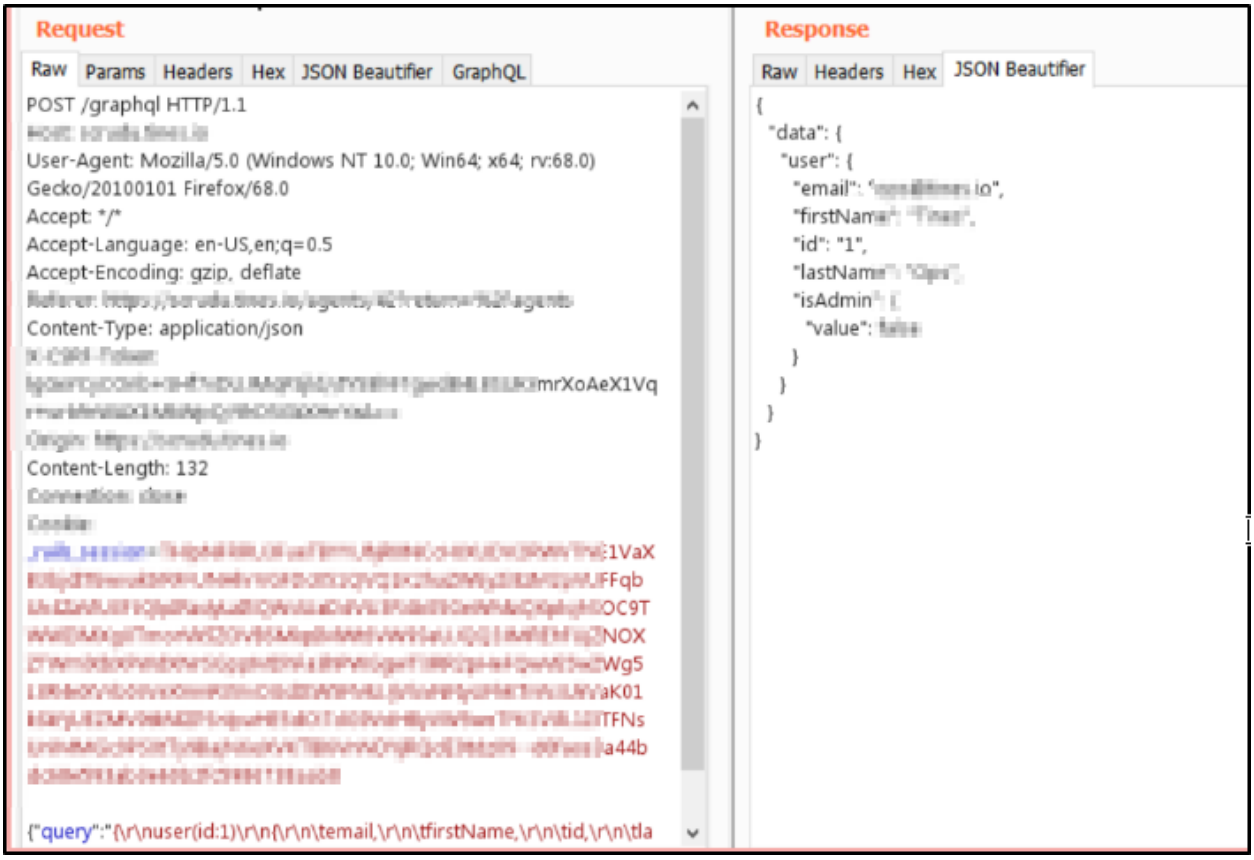
See an OTP ? or MFA code check for rate limit

3) SQL injection

Yes you read it right, GraphQL does not mean that app don't have the basic bugs so always check for it. But I've noticed alot of the GraphQL APIs are now connected to MongoDB so you need to familiarize your self NO SQL injection. (yes that's a thing)
4) CSRF
5) Nested Graphql queries for DOS (https://voidsec.com/graphql-security-overview-and-testing-tips/)

and any other API bug that you can image exists in GraphQL.

Using this knowledge I was able to view user PII in a web app that only admins were allowed to view and I scored a small bounty with it just a few hundred $



Well thats about it, if there is anything I missed or typed wrong feel free to hit me up on
https://twitter.com/th3_3inst3in