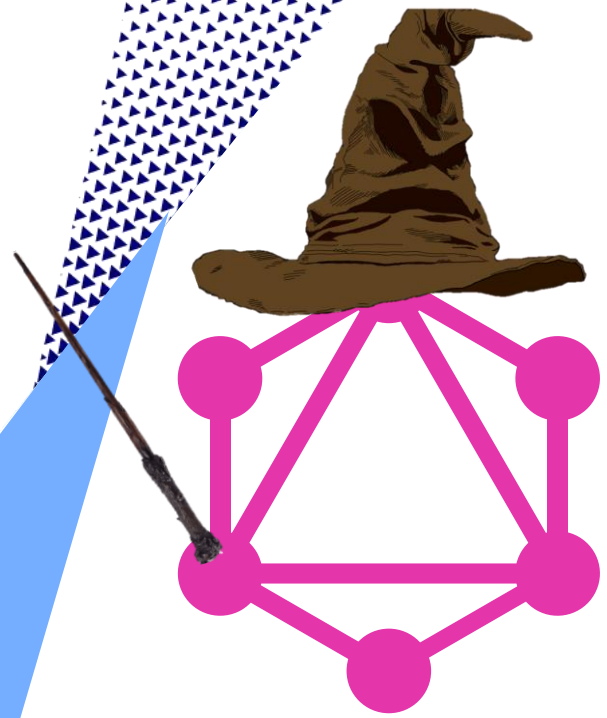
CyberCX

Cyber Security + Customer Experience



Fantastic GraphQL Bugs and Where To Find Them

Thank You to Our Sponsors and Hosts!

OWASP NEW ZEALAND
owasp.org.nz

QUANTUM SECURITY

AUT

CyberCX

DATACOM

snyk

Auth0

Checkmarx

HCL AppScan

kordia

LATERAL SECURITY

MICRO FOCUS

Pulse Security
www.pulsesecurity.co.nz

RedShield

Flux

SEQA
Information Security

Cobalt

LACEWORK

SecureFlag

Without them, OWASP New Zealand Day couldn't happen

whoami

- Security Consultant @ CyberCX
- Been working in the industry for 4 years

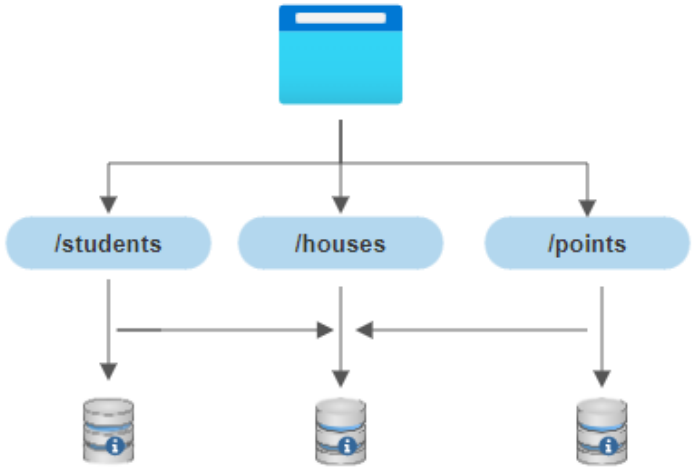
What is GraphQL

- GraphQL is a query language for APIs
- Allows you to query data as well as perform CRUD operations via a single endpoint
- Schema based
- GraphQL is designed for efficiency, you retrieve only what you need, nothing more nothing less

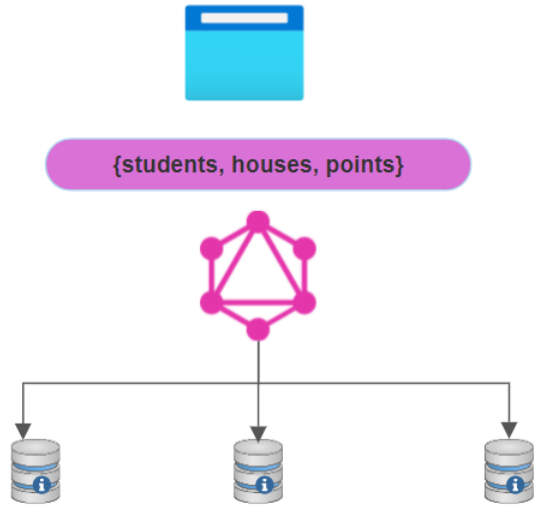
https://apievangelist.com/2018/06/29/graphql-and-rest-differences-explained-with-burgers/

REST vs GraphQL

REST



GraphQL



Queries and Mutations

- Query = GET

```
query {
  allPosts {
    edges {
      node {
        title
        body
        users
          { username }
        }
      }
    }
}
```


```
{
  "data": {
    "allPosts": {
      "edges": [
        {
          "node": {
            "title": "Yer a wizard Harry!",
            "body": "Yer gonna fly a broom, learn spells n shit. And yer gonna enjoy it",
            "users": {
              "username": "iliekdragons"
            }
          }
        },
        {
          "node": {
            "title": "HARRY DID YA PUT YA NAME IN DA GOBLET OF FIYAHH",
            "body": "said Dumbledore calmly",
            "users": {
              "username": "dumbledore"
            }
          }
        },
        {
          "node": {
            "title": "Pottah!",
            "body": "My father will hear about this",
            "users": {
              "username": "draco malfoy"
            }
          }
        }
      ]
    }
  }
}
```


- Mutation = POST, DELETE

```
mutation{
  userMutation(id:1, password:"AlasEarwax!"){
    id
    password
    updated_at
  }
}
```


```
{
  "data":{
    "userMutation":{
      "id":1,
      "password":"AlasEarWax!",
      "updated_at":"2022-03-02 23:02:22"
    }
  }
}
```

Accio Vulnerability

- IDOR
- SQLi
- Broken Access Control
- DoS

Tools?

- Burp Plugins – inQL and GraphQL Raider – helps formatting GraphQL in Burp
- GraphQL Voyager – Mapping out the schema
- Insomnia[.]rest – API Client
- GraphQL 😏

Introspection

- Allows you to retrieve the API schema

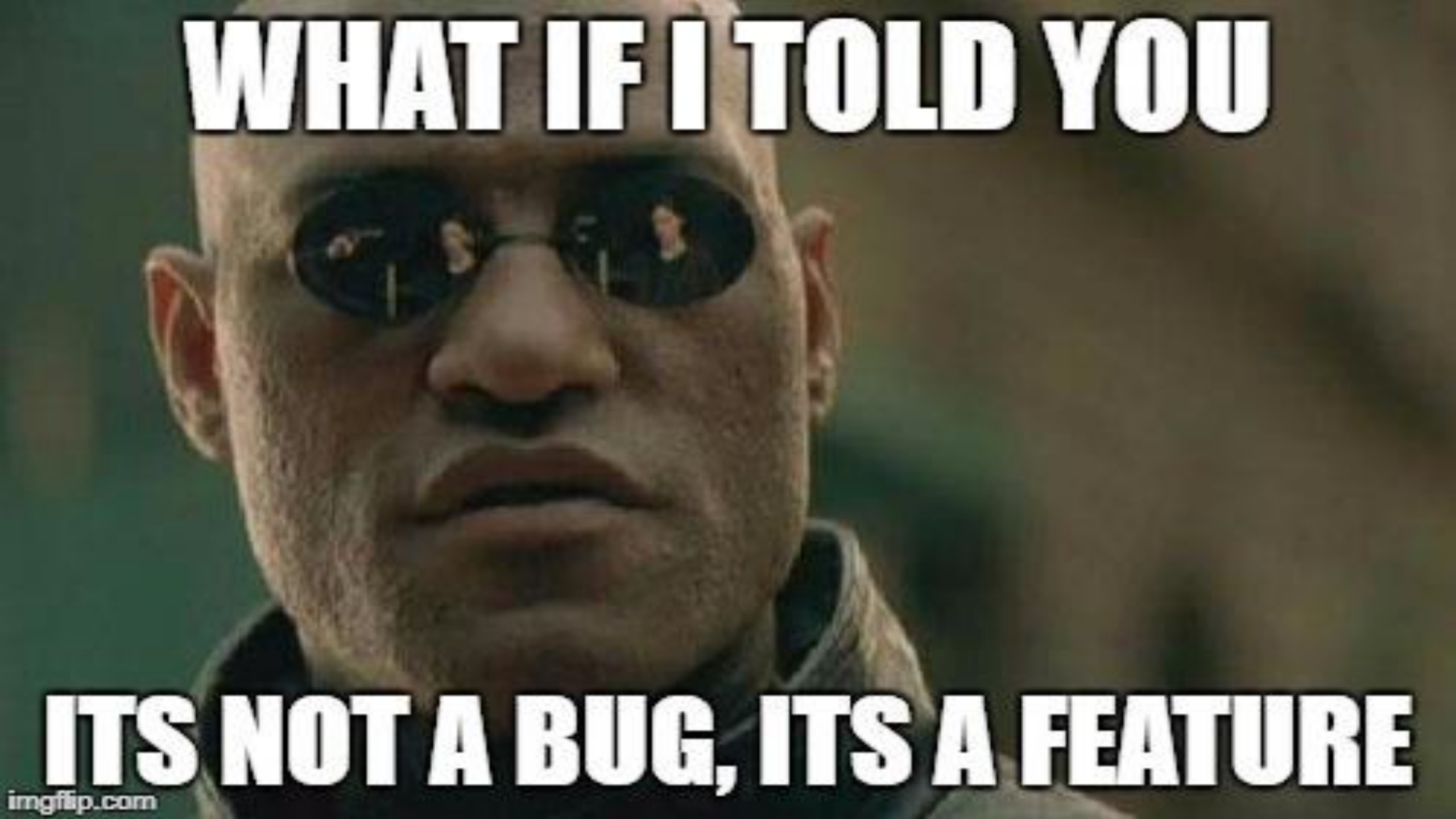
```
query MyQuery {
  __schema {
    types {
      name
      fields {
       name
      }

    }
  }
}
```


```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Query",
          "fields": [
            {
              "name": "node"
            },
            {
              "name": "allPosts"
            },
            {
              "name": "allUsers"
            }
          ]
        },
        {
          "name": "Node",
          "fields": [
            {
              "name": "id"
```


Look for:

- /v1/explorer
- /v1/graphiql
- /graph
- /graphql
- /graphql/console/
- /graphql.php
- /graphiql
- /graphiql.php

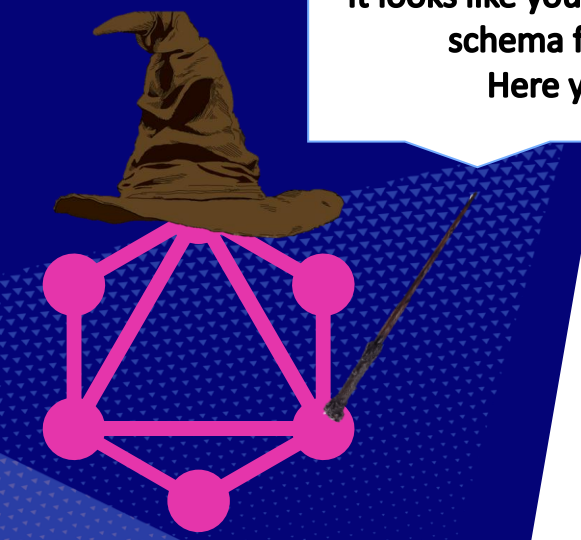

WHAT IF I TOLD YOU
ITS NOT A BUG, ITS A FEATURE
imgflip.com

Introspection

- Introspection Query

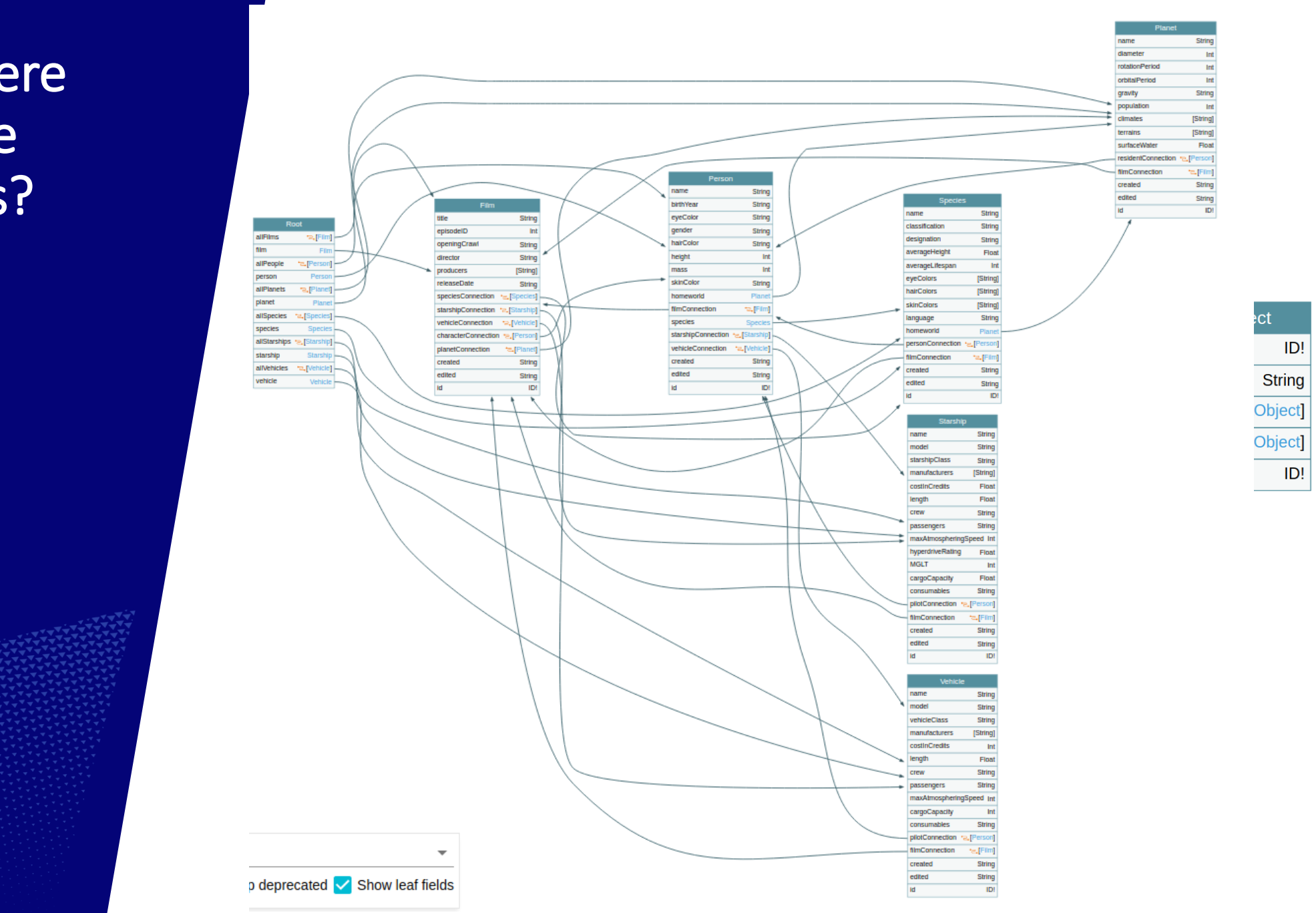
```
{__schema{queryType{name}mutationType{name}subscriptionType{name}types{...FullType}directives{name
description locations args{...InputValue}}}}fragment FullType on __Type{kind name description
fields(includeDeprecated:true){name description args{...InputValue}type{...TypeRef}isDeprecated
deprecationReason}inputFields{...InputValue}interfaces{...TypeRef}enumV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

It looks like you want request the schema for this API.
Here you go ☺




```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": null,
      "subscriptionType": null,
      "types": [
        {
          "kind": "OBJECT",
          "name": "Query",
          "description": null,
          "fields": [
            {
              "name": "node",
              "description": "The ID of the object",
              "args": [
                {
                  "name": "id",
                  "description": null,
                  "type": {
                    "kind": "NON_NULL",
                    "name": null,
                    "ofType": {
                      "kind": "SCALAR",
                      "name": "ID",
                      "ofType": null
                    }
```

So where are the graphs?



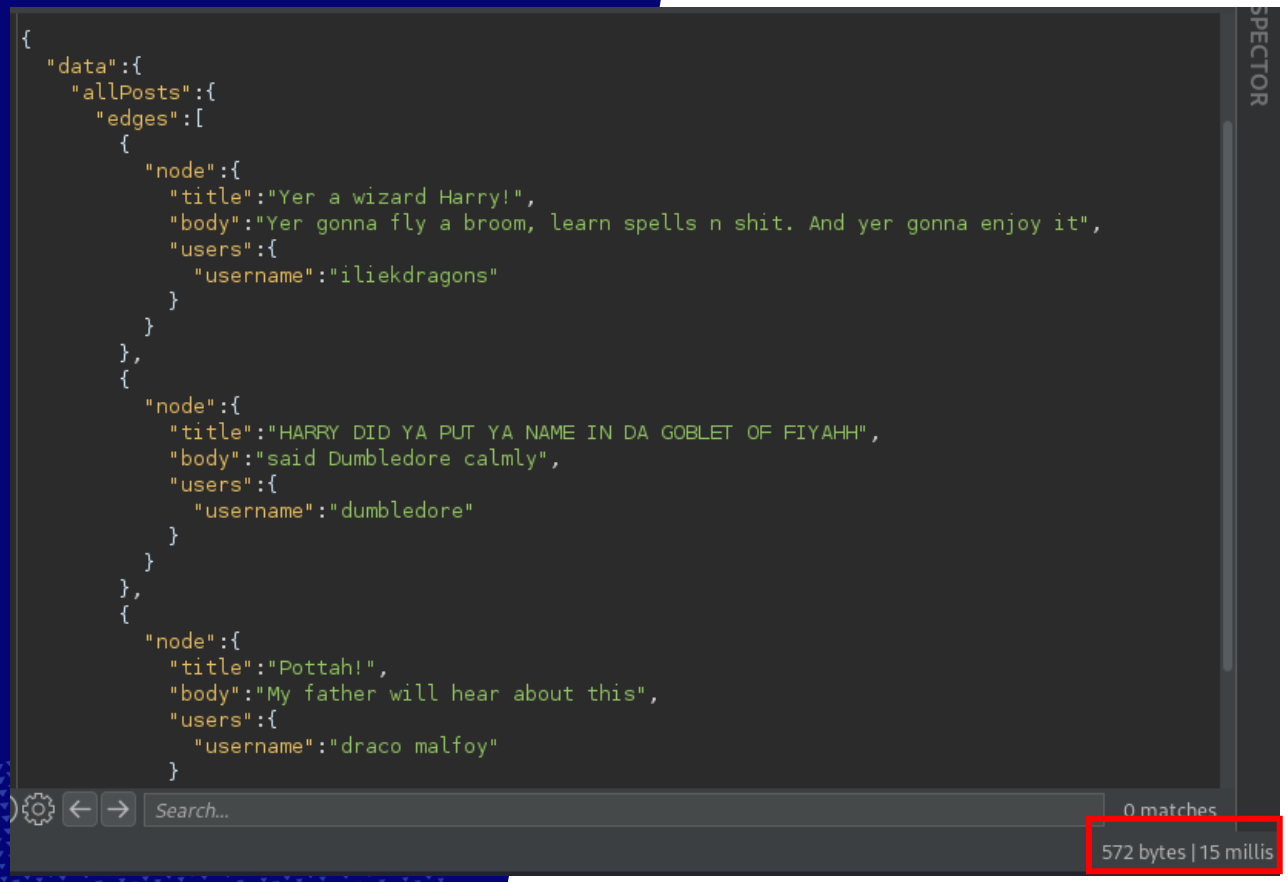
DoS

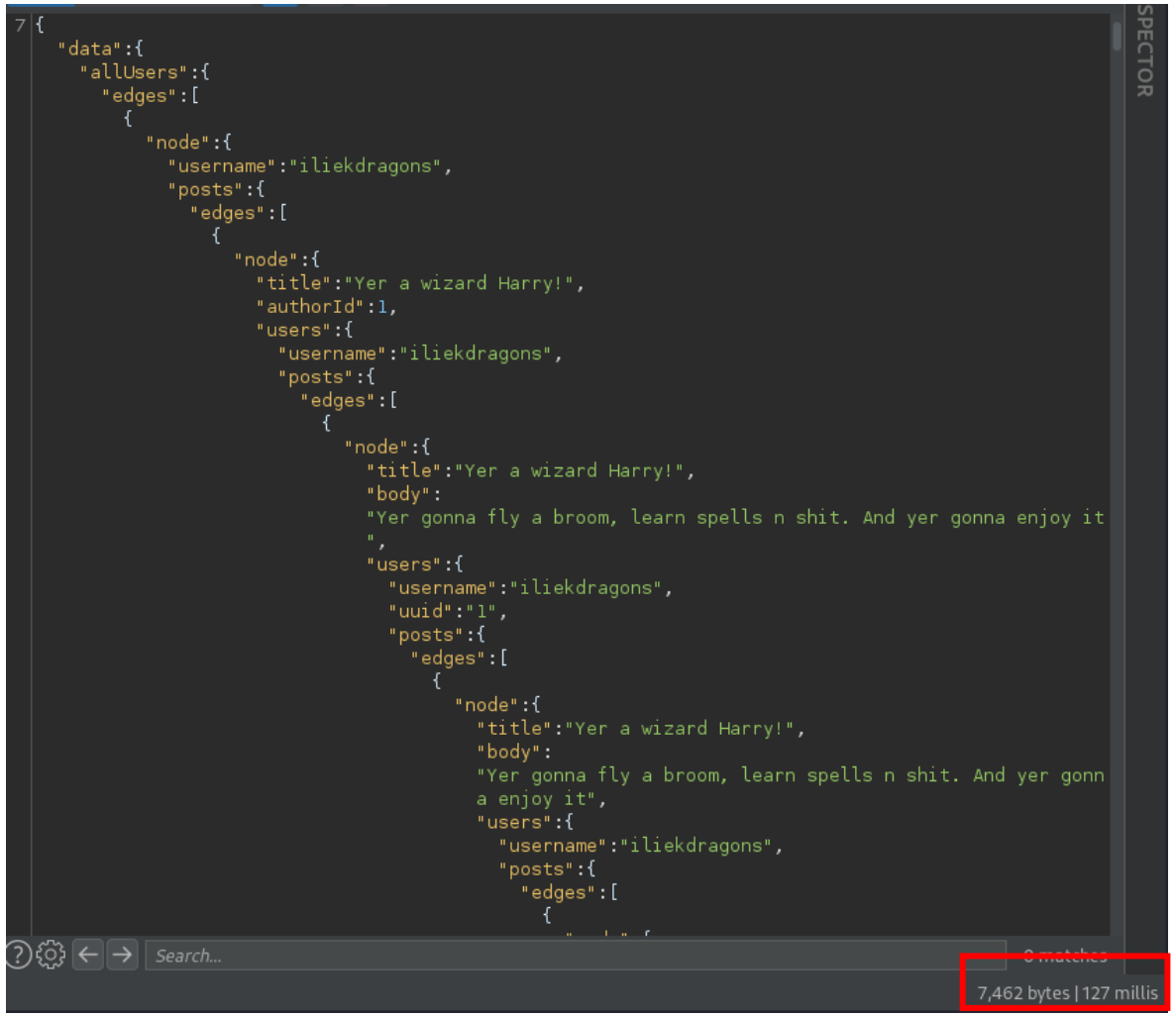
- As queries can be very complex - you can nest them and call them recursively
- E.g In a blog, you will have Users and Posts. A user can have multiple posts and each post will have a user.
- We can call this recursively.

Query Variables Injection Points

```
{
  allUsers {
    edges {
      node {
        username
        posts {
          edges {
            node {
              title
              authorId
              users {
                username
                posts {
                  edges {
                    node {
                      title
                      body
                      users {
                        username
                        uuid
                        username
                        uuid
                        posts {
                          edges {
                            node {
                              title
                              body
                              users {
                                username
                                posts {
                                  edges {
                                    node {
                                      title
                                      body
```

DoS



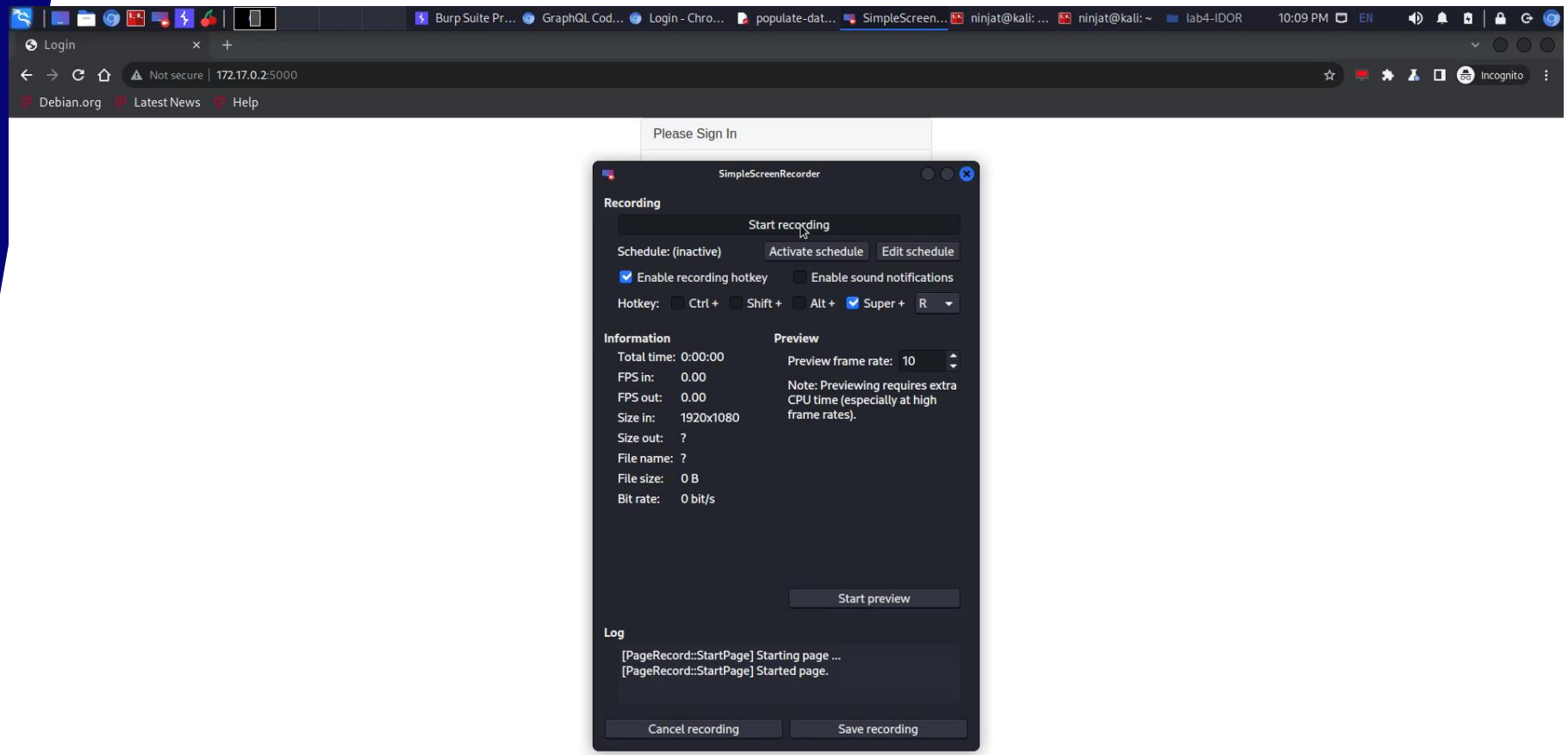


DoS Mitigation?

- Limit Maximum Query Depth
- Throttling Based on Server Time or Query Complexity
- Audit your query before production

- https://www.npmjs.com/package/graphql-validation-complexity
- https://github.com/4Catalyzer/graphql-validation-complexity
- https://github.com/slicknode/graphql-query-complexity

IDOR

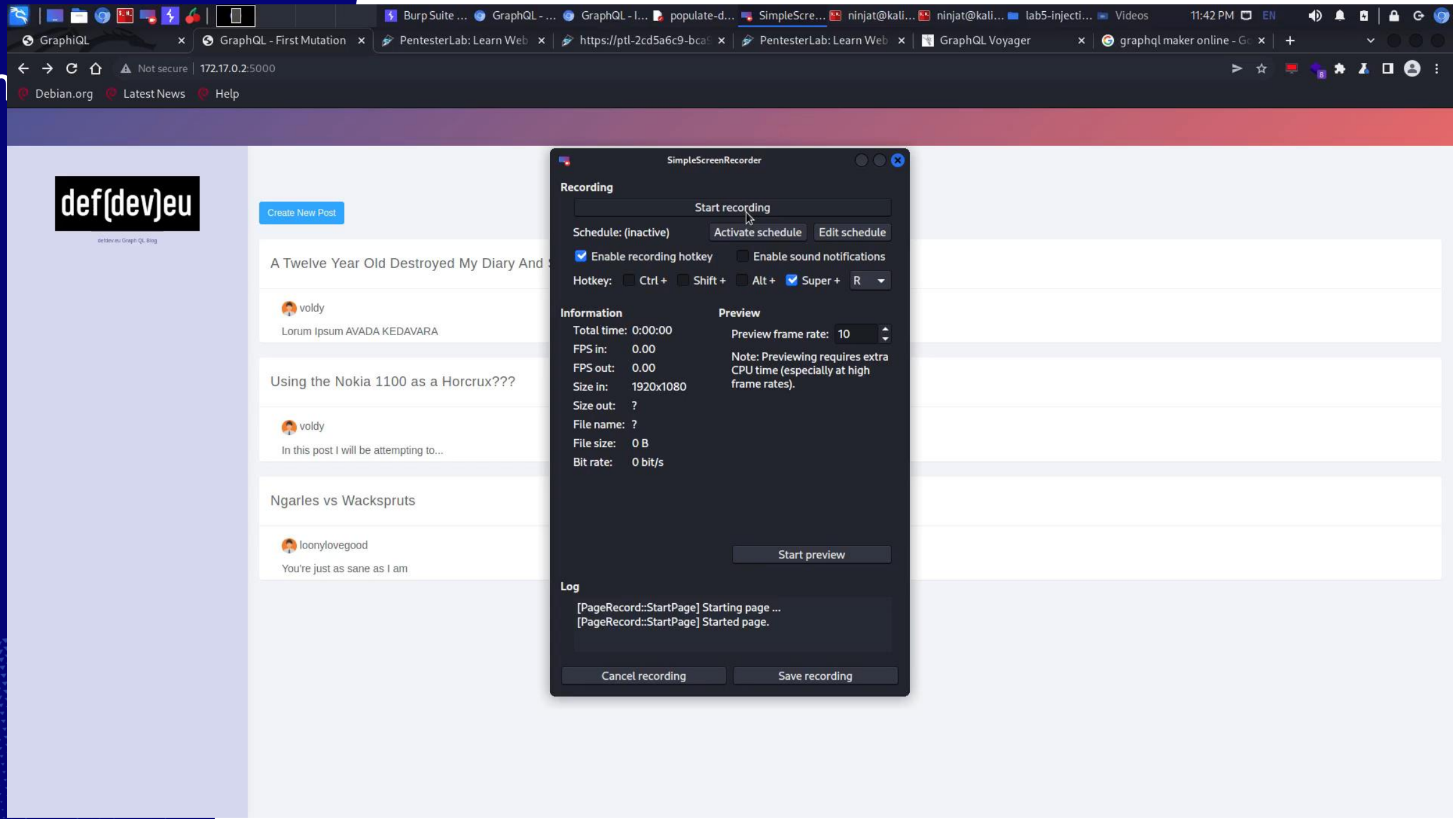


IDOR Mitigation?

- Use GUIDs instead of IDs
  - Increases complexity

- Implement Role Based Access Controls (RBACs)
  - Validate that the requested is authorised to perform the action

In



Injection Mitigation?

- Input Validation
  - Whitelist approach
- Use parameterised queries

Examples

- Testing a crowdfunding style app
- Access controls weren’t applied to GraphQL endpoint
- Allowed **unauthed** payments to funding requests, remove funding from one project to another and resetting user passwords
- Create your own project -> Move funding to your own project -> $$$$

Q&A

CyberCX
New Zealand