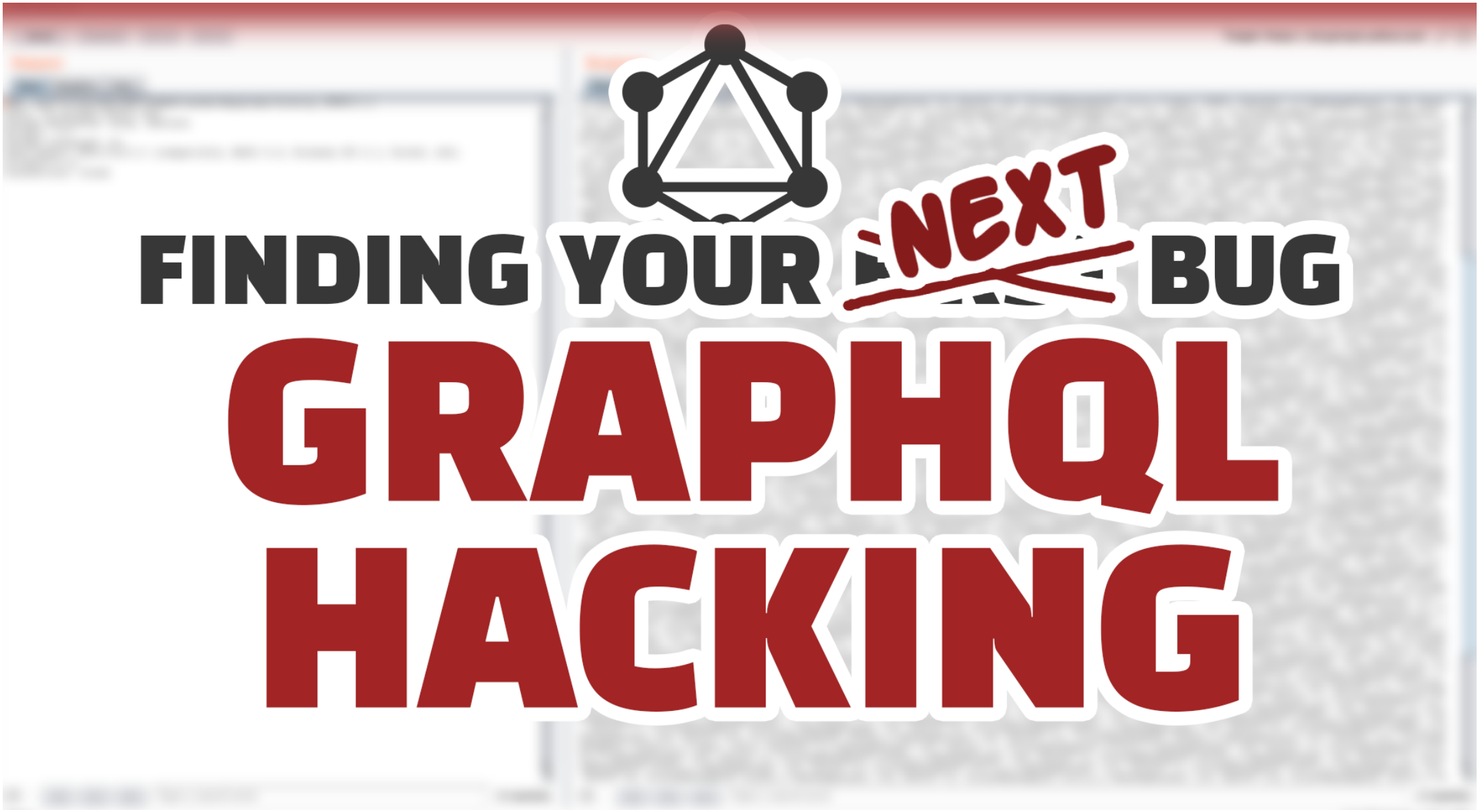
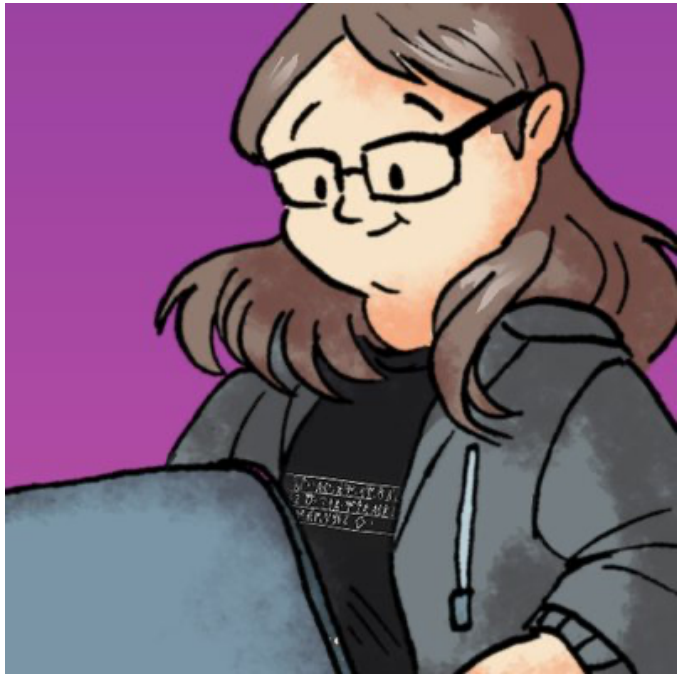

FINDING YOUR NEXT BUG
GRAPHQL
HACKING

About me

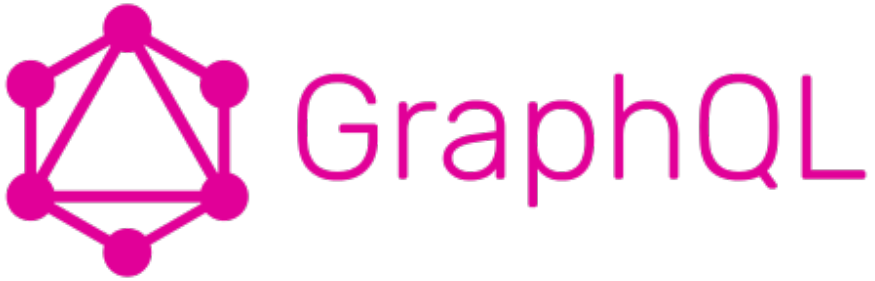


What is GraphQL?

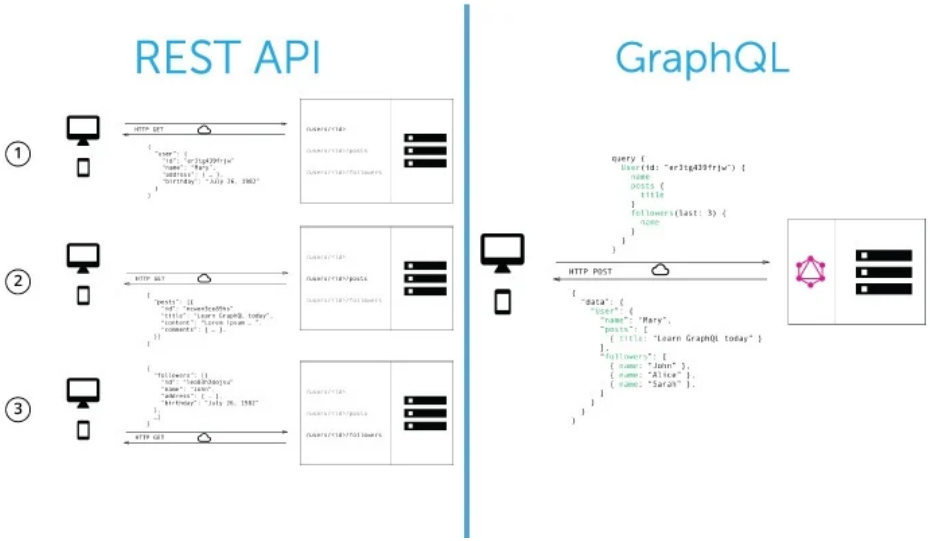
I’ve talked about GraphQL a lot but what is it exactly?

GraphQL is an API

- But not quite as we know it
- Usually when developers write an API they write a lot of endpoints
  - GET /resource/1
  - GET /resource/
  - POST /resource/
- This ends up as several endpoints, one for each CRUD operation per resource, eg users, posts, comments….
- But GraphQL is special, we only use 1 endpoint for every resource



Why one endpoint?

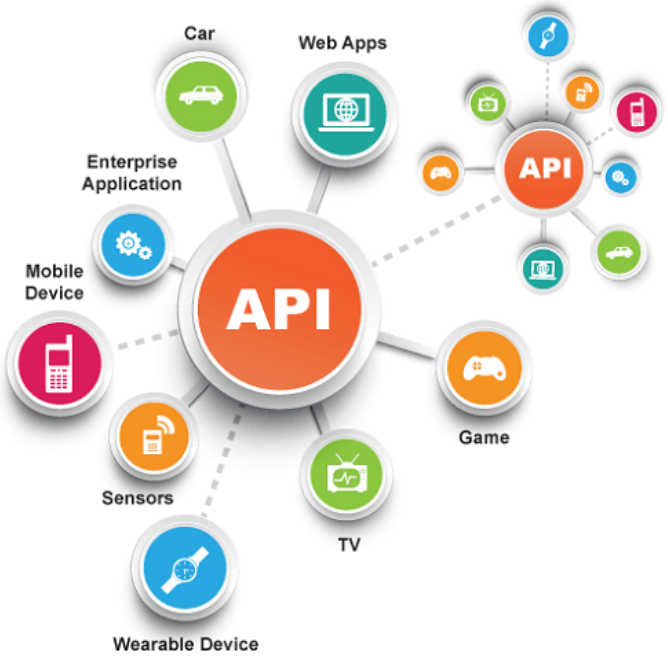


- This makes, on the dev side, things quicker to make
- Instead of making endpoints, they write a few queries for common operations
- Then they use these queries in lots of different places
- Very different to SQL queries in syntax but not in style!

Why GraphQL?

- It’s a newer technology
- Developers may be adopting it without being fully aware of the security concerns
- Which is not great for their users, but good for us as bug hunters!
- It’s intimidating but not that much harder than API hacking in general
- You also don’t need to do recon in a lot of cases (more to come…)

Where to find GraphQL?



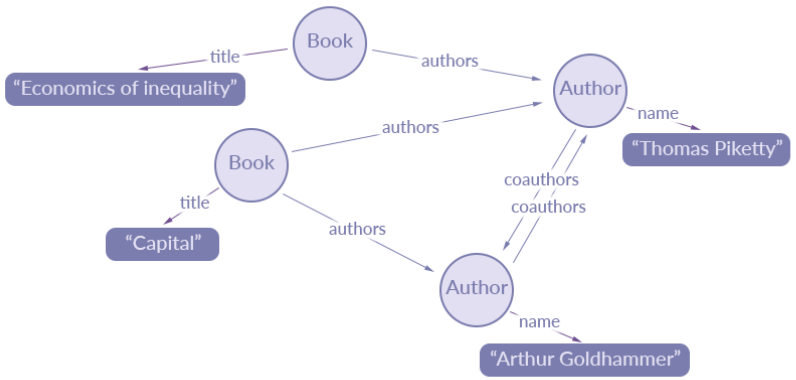
- Same place you find other APIs
- Tends to be more common with newer applications
- Yahoo, Shopify, HackerOne all use GraphQL
- GraphQL is usually located at specific endpoints
  - qql, graphql, graphiql, graphql/console
  - But look out for requests/responses referencing: queries, mutations

How does Graph QL work?

- GraphQL implements a **graph structure** as the database
- There are **queries** and **mutations**
- **Queries** fetch data
- **Mutations** allow the data to be edited
- **Fragments** allow for easily saved lists of fields
- **Metafields** allow for the inspection of query or mutation information

Graph Structure

- Usually we represent data as tables, spreadsheets of data
- But we often link 2 different tables together using IDs
- Now we can still think of it in flat structures
- But the natural form of this data is a graph!



Queries

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
```

```
{
  human(id: "1000") {
    name
    height
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 1.72
    }
  }
}
```

```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

- Queries in GraphQL are written to be flexible
- You can request any field easily, including related entities on the graph, and then a field associated with that entity
- Written as functions
  - Can return a single result or several
  - Can also include arguments
  - Or data manipulation
- Very structured

Mutations

- While queries fetch data, a mutation edits it
- Can be an edit with assigned variables or deleting
- Can also fetch data after modifying it

Fragments and Metaqueries

```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
```

```
{
  search(text: "an") {
    __typename
    ... on Human {
      name
    }
    ... on Droid {
      name
    }
    ... on Starship {
      name
    }
  }
}
```

```
{
  "data": {
    "search": [
      {
        "__typename": "Human",
        "name": "Han Solo"
      },
      {
        "__typename": "Human",
        "name": "Leia Organa"
      },
      {
        "__typename": "Starship",
        "name": "TIE Advanced x1"
      }
    ]
```

- Fragments allow you to decide a list of fields and request multiple queries use the same list
- Used for comparisons
- Metafields allow you to inspect the API, __typename returns the typname
- Extremely important for introspection queries!

Okay but how?

- How are you supposed to know???
- What queries are written?
- What mutations there are?
- What the arguments are?
- What fields you can return

- I’m glad you asked, remember those Fragments….

Introspection & Recon

How to find all those queries and mutations

Why is it important to learn GQL syntax?

- **GraphQL is only challenging because of it's syntax not because it has really hard bugs**
- And so it's important to understand how queries and mutations are written and concepts like metaqueries and fragments
- Why? GraphQL can be REALLY easy to do recon on!
- How? Introspection

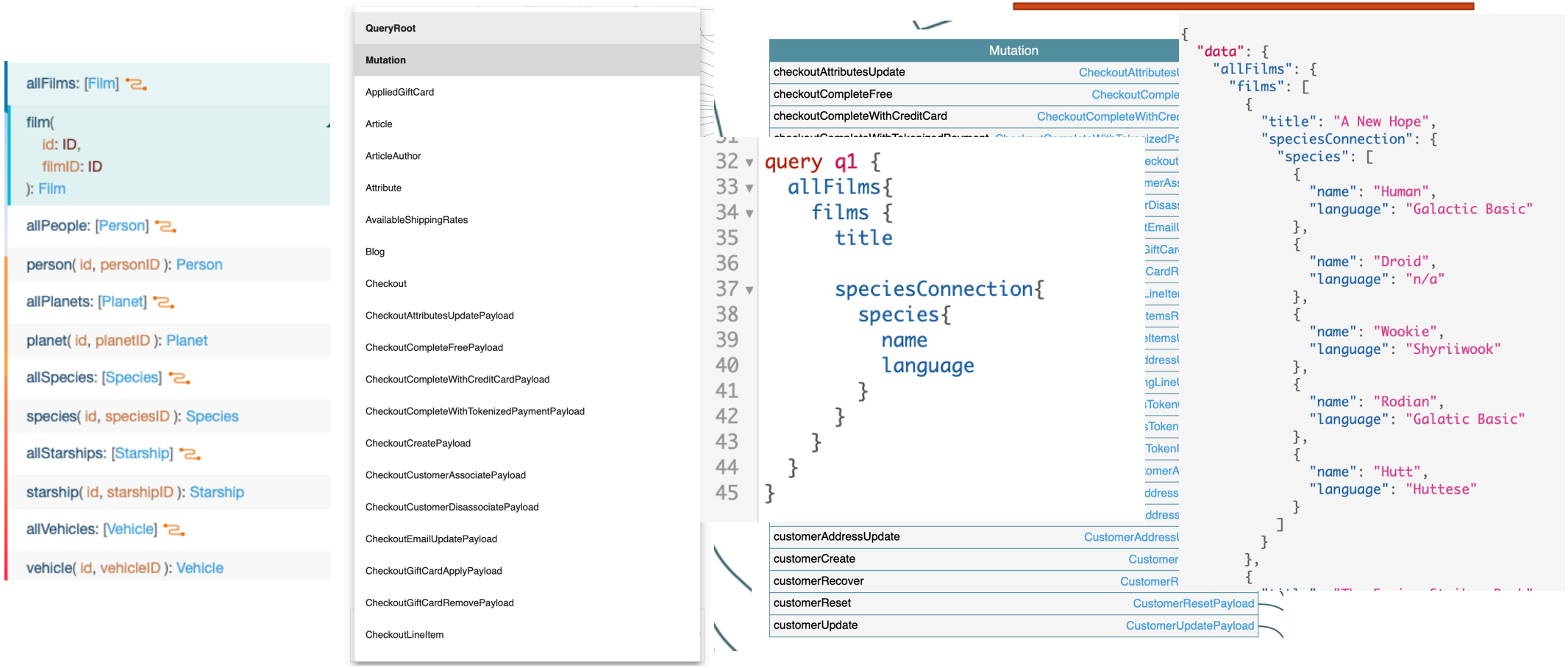
```
query q1 {
  allFilms {
    films {
      director

      starshipConnection {
        starships {
          name
          pilotConnection{
            pilots{
              name
            }
          }
        }
      }
    }
  }
}
```

Introspection

fragment+FullType+on+__Type+{++kind++name++description++fields(includeDeprecated%3a+true)+{++++name++++description++++args+{++++++...InputValue++++}++++type+{++++++...TypeRef++++}++++isDeprecated++++deprecationReason++}++inputFields+{++++...InputValue++}++interfaces+{++++...TypeRef++}++enumValues(includeDeprecated%3a+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++++++++++++++}++++++++++++}++++++++++}++++++++}++++++}++++}++}}query+IntrospectionQuery+{++__schema+{++++queryType+{++++++name++++}++++mutationType+{++++++name++++}++++types+{++++++...FullType++++}++++directives+{++++++name++++++description++++++locations++++++args+{++++++++...InputValue++++++}++++}++}}

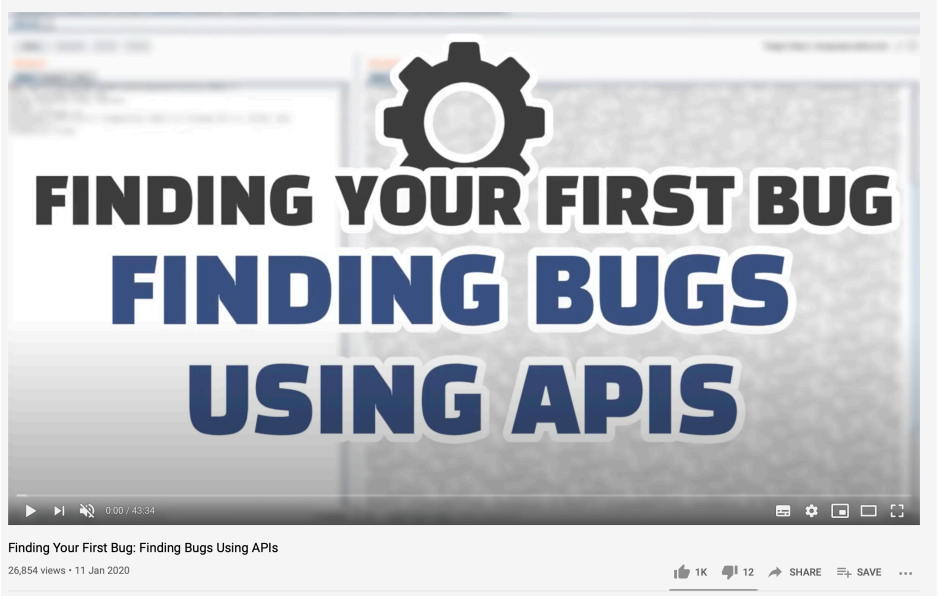
_schema{queryType{name},mutationType{name},types{kind,name,description,fields(includeDeprecated:true){name,description,args{name,description,type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}}}},defaultValue},type{kind,name,ofType{kind,name,ofType{kind,name,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,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,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,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}}}

GraphQL voyager



So what next?

- Now we know all the endpoints we actually skip right to the API hacking!
- Note: Introspection itself is not usually considered a bug, usually it’s considered documentation
- We want to test all these queries just like we’d test a regular API
- But what if introspection has been disabled?



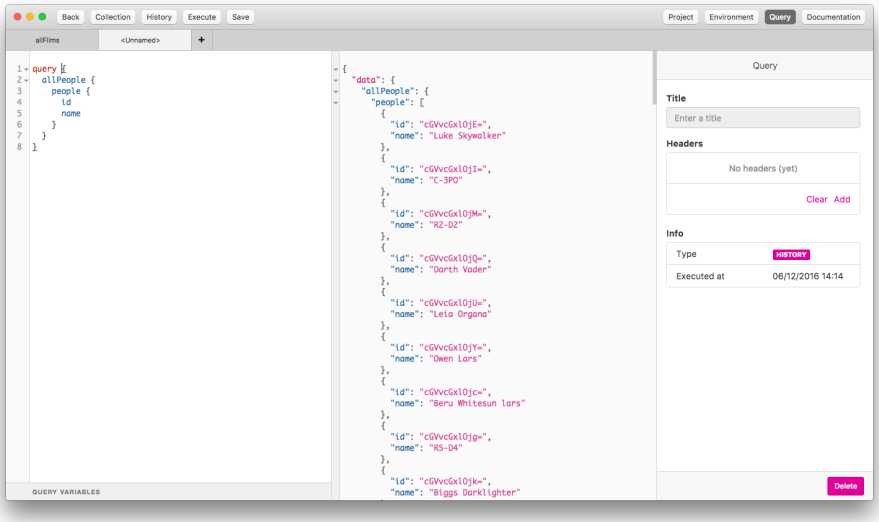
How to do more traditional recon

- Even if introspection is turned off we can still do recon to find API endpoints
- Click buttons, figure out the structure of queries, replace likely entity names
  - Or just test visible gql endpoints!
- Keep an eye out for:
  - Any documentation, this is likely to list Gql endpoints
  - Errors, Gql might even tell you what you messed up!

GraphQL tools

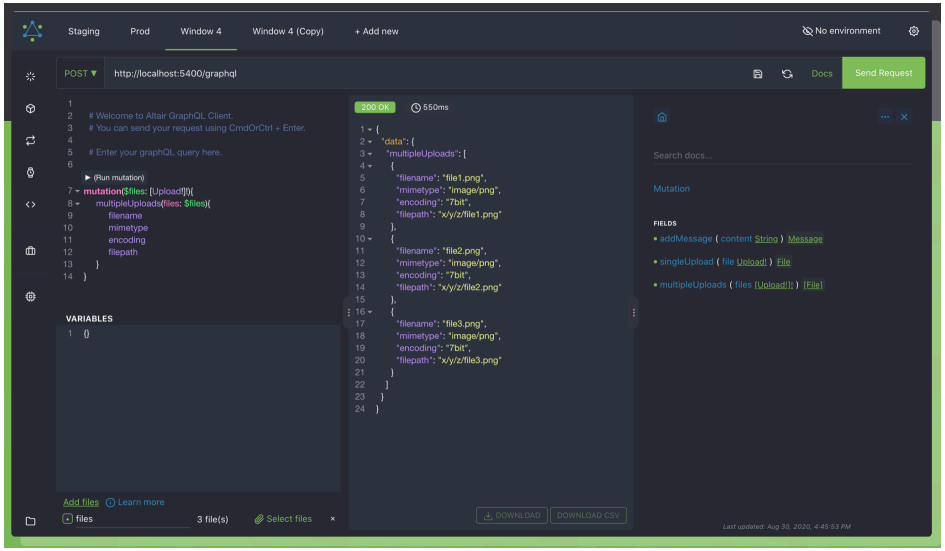
There are some tools which are unique to GraphQL, here’s a list

GraphQL IDE – GraphQL IDE



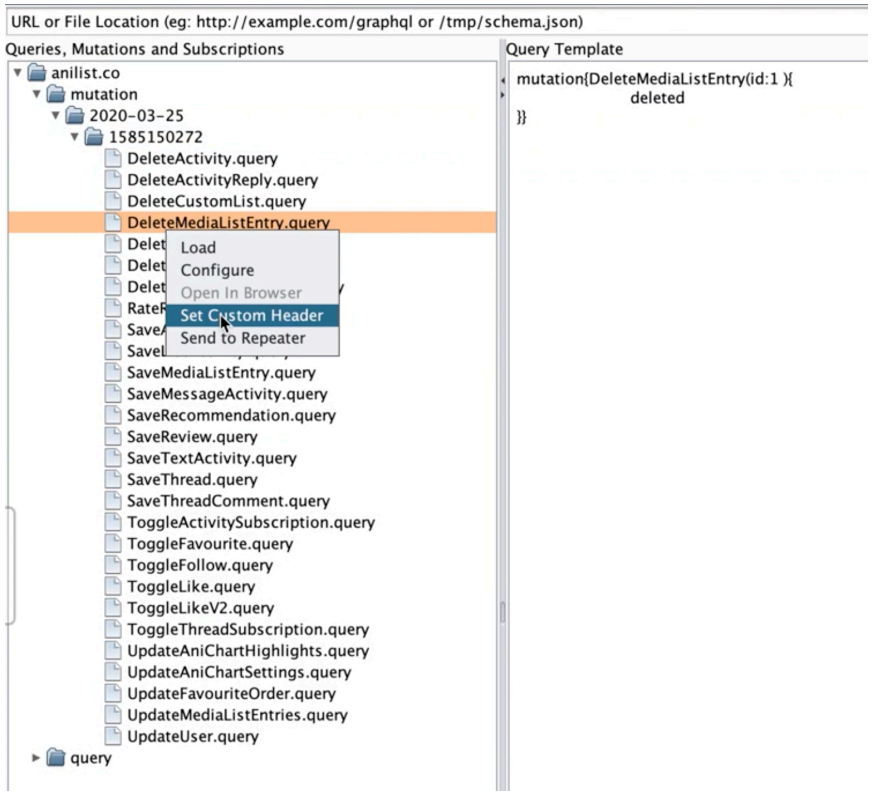
- Makes it a little easier to interact with a graphQL API by creating an IDE
- Fairly easy to use, you're still going to be writing queries by hand
- For when you're in the exploitation stage

Altair – GraphQL IDE



- Makes it a little easier to interact with a graphQL API by creating an IDE
- Similar to Postman in design, so if you're more used to Postman might be worth looking in Altair
- You'll still be hand writing queries though!

InQL – Burp addon and scanner



- Burp addon
- Send an introspection query to show the queries and mutations
- Integrates with Burp allowing you to send to repeater
- Nice for Burp power users who already use a lot of tools via Burp
- Can also generate documentation!

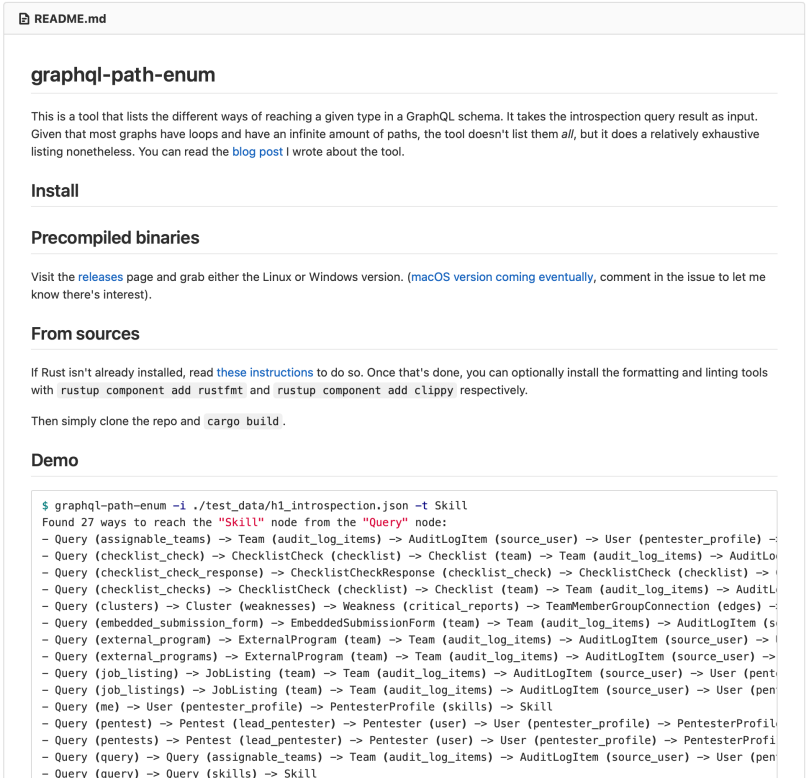
GraphQL Map

```
$ git clone https://github.com/swisskyrepo/GraphQLmap
$ python graphqlmap.py
```

```
                                    Author:Swissky Version:1.0
usage: graphqlmap.py [-h] [-u URL] [-v [VERBOSITY]] [--method [METHOD]] [--headers [HEADERS]]

optional arguments:
  -h, --help          show this help message and exit
  -u URL              URL to query : example.com/graphql?query={}
  -v [VERBOSITY]      Enable verbosity
  --method [METHOD]   HTTP Method to use interact with /graphql endpoint
  --headers [HEADERS] HTTP Headers sent to /graphql endpoint
  --json              Send requests using POST and JSON
```

- Command line tool for introspection and querying
- If you’re more a command line user, this will probably suit you well
- Can send queries but also generate nosql and sql injection payloads

graphql-path-enum – Understand large graphs



README.md

graphql-path-enum

This is a tool that lists the different ways of reaching a given type in a GraphQL schema. It takes the introspection query result as input. Given that most graphs have loops and have an infinite amount of paths, the tool doesn't list them *all*, but it does a relatively exhaustive listing nonetheless. You can read the blog post I wrote about the tool.

Install

Precompiled binaries

Visit the releases page and grab either the Linux or Windows version. (macOS version coming eventually, comment in the issue to let me know there's interest).

From sources

If Rust isn't already installed, read these instructions to do so. Once that's done, you can optionally install the formatting and linting tools with `rustup component add rustfmt` and `rustup component add clippy` respectively.

Then simply clone the repo and `cargo build`.

Demo

```
$ graphql-path-enum -i ./test_data/h1_introspection.json -t Skill
Found 27 ways to reach the "Skill" node from the "Query" node:
- Query (assignable_teams) -> Team (audit_log_items) -> AuditLogItem (source_user) -> User (pentester_profile) -
- Query (checklist_check) -> ChecklistCheck (checklist) -> Checklist (team) -> Team (audit_log_items) -> AuditLo
- Query (checklist_check_response) -> ChecklistCheckResponse (checklist_check) -> ChecklistCheck (checklist) -> (
- Query (checklist_checks) -> ChecklistCheck (checklist) -> Checklist (team) -> Team (audit_log_items) -> AuditL
- Query (clusters) -> Cluster (weaknesses) -> Weakness (critical_reports) -> TeamMemberGroupConnection (edges) -
- Query (embedded_submission_form) -> EmbeddedSubmissionForm (team) -> Team (audit_log_items) -> AuditLogItem (s
- Query (external_program) -> ExternalProgram (team) -> Team (audit_log_items) -> AuditLogItem (source_user) -> (
- Query (external_programs) -> ExternalProgram (team) -> Team (audit_log_items) -> AuditLogItem (source_user) ->
- Query (job_listing) -> JobListing (team) -> Team (audit_log_items) -> AuditLogItem (source_user) -> User (pent
- Query (job_listings) -> JobListing (team) -> Team (audit_log_items) -> AuditLogItem (source_user) -> User (pen
- Query (me) -> User (pentester_profile) -> PentesterProfile (skills) -> Skill
- Query (pentest) -> Pentest (lead_pentester) -> Pentester (user) -> User (pentester_profile) -> PentesterProfil
- Query (pentests) -> Pentest (lead_pentester) -> Pentester (user) -> User (pentester_profile) -> PentesterProfi
- Query (query) -> Query (assignable_teams) -> Team (audit_log_items) -> AuditLogItem (source_user) -> User (pen
- Query (query) -> Query (skills) -> Skill
```

- Because of the interconnected nature of GQL and how complex schemas can get, it can be difficult to work out how to get from Node A->Node B
- This tool walks an introspection query result to show how two entities are related

GraphQL Bugs in the Wild

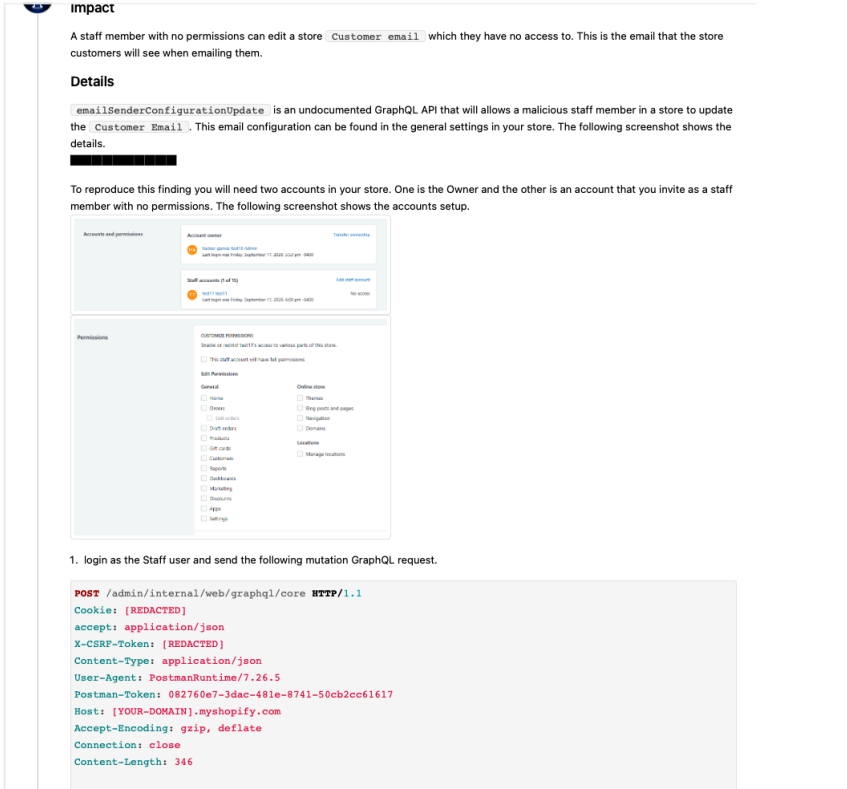
What kind of bugs should we be looking for?

Common GraphQL Bugs

- The same as API bugs!
- The usual suspects
  - Information disclosure
  - IDORs
  - Bypassing client restrictions (eg WAF for XSS)
- What makes Gql special? The syntax!
  - The bugs are the same!!

A staff member with no permissions can ed it Store Customer Email - $1,500

- Undocumented GraphQL API endpoint
- Allowed you to change a store's customer email
- This is the email customers see when emailing a store
- Impact is reputational for the store
- Very simple IDOR



Impact

A staff member with no permissions can edit a store `Customer email` which they have no access to. This is the email that the store customers will see when emailing them.

Details

`emailSenderConfigurationUpdate` is an undocumented GraphQL API that will allows a malicious staff member in a store to update the `Customer Email`. This email configuration can be found in the general settings in your store. The following screenshot shows the details.

[redacted]

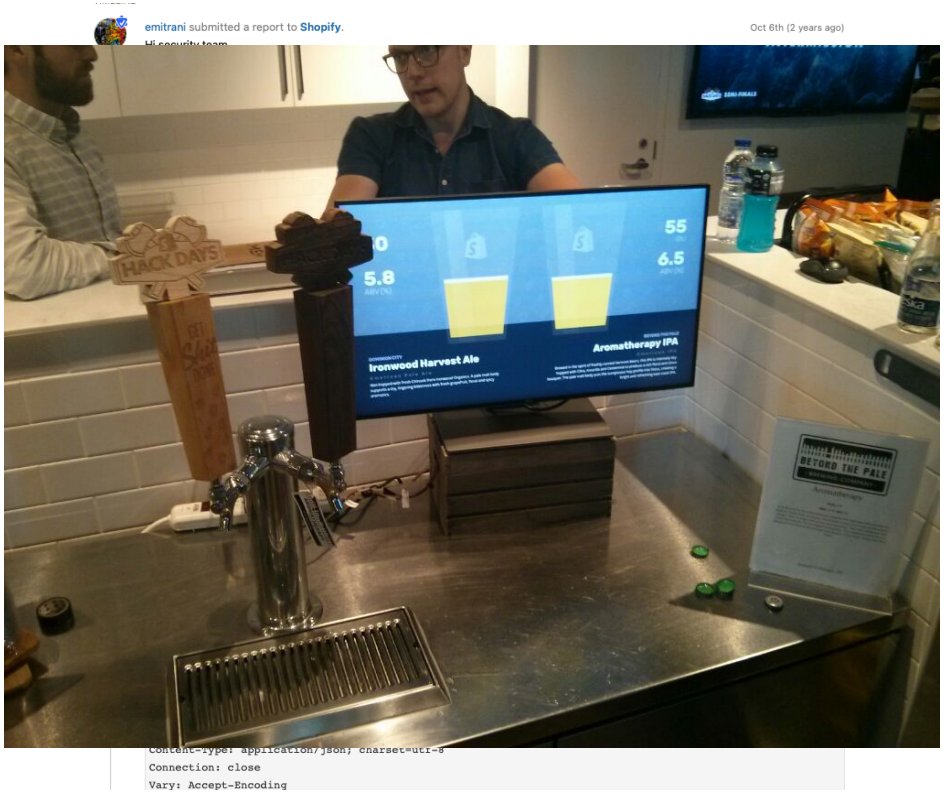
To reproduce this finding you will need two accounts in your store. One is the Owner and the other is an account that you invite as a staff member with no permissions. The following screenshot shows the accounts setup.

1. login as the Staff user and send the following mutation GraphQL request.

```
POST /admin/internal/web/graphql/core HTTP/1.1
Cookie: [REDACTED]
accept: application/json
X-CSRF-Token: [REDACTED]
Content-Type: application/json
User-Agent: PostmanRuntime/7.26.5
Postman-Token: 082760e7-3dac-481e-8741-50cb2cc61617
Host: [YOUR-DOMAIN].myshopify.com
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 346
```

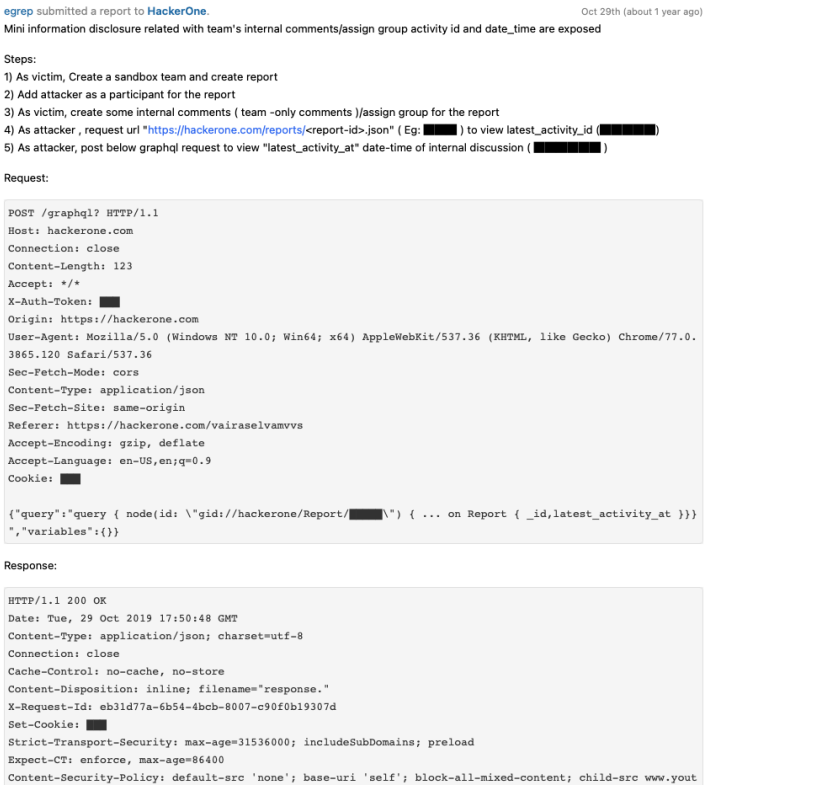
H1514 [beerify.shopifycloud.com] GraphQL discloses internal beer consumption - $802.20

- Find all subdomains of *.shopifycloud.com, and send a random query to /graphql filter out bad requests
- Use introspection to find out what queries you can run on any endpoint
- Realise it includes references to beer
- Send crafted queries to find out what is possible to discover
- Realise that this should be internal only (office beer consumption)
- Very simple information disclosure



latest_activity_id and latest_activity_at may disclose information about internal activities to unauthorized users - $1,000

- By querying the GraphQL API directly it was possible for a user to have some information about internal activity on a HackerOne report
- By realising the “latest_activity_id” and “latest_activity_at” returned some info information on internal comments
- Information disclosure again!



egrep submitted a report to HackerOne. Oct 29th (about 1 year ago)

Mini information disclosure related with team's internal comments/assign group activity id and date_time are exposed

Steps:
1) As victim, Create a sandbox team and create report
2) Add attacker as a participant for the report
3) As victim, create some internal comments (team -only comments)/assign group for the report
4) As attacker , request url "https://hackerone.com/reports/<report-id>.json" (Eg: [redacted]) to view latest_activity_id ([redacted])
5) As attacker, post below graphql request to view "latest_activity_at" date-time of internal discussion ([redacted])

Request:

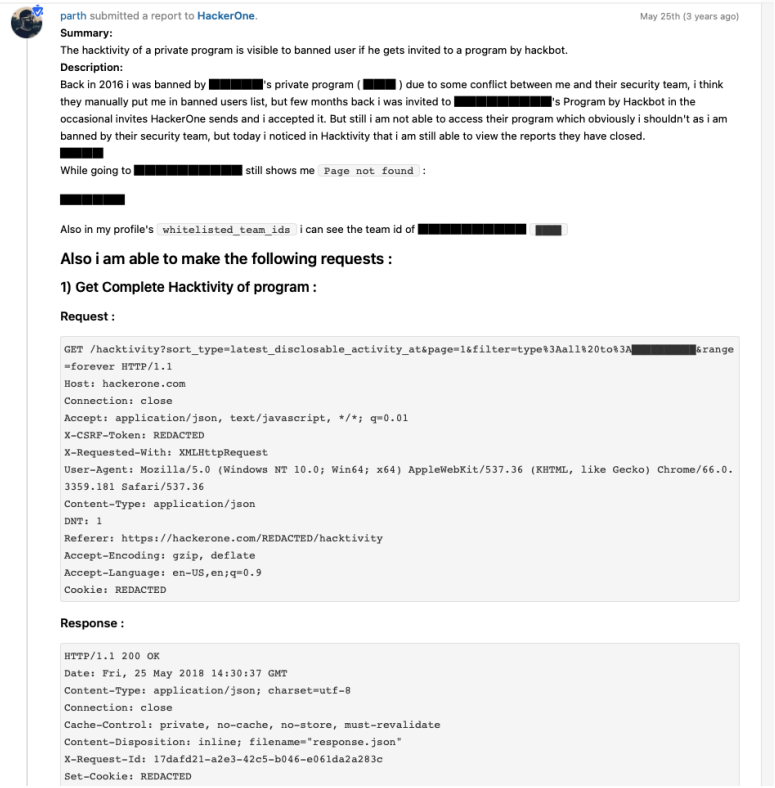
```
POST /graphql? HTTP/1.1
Host: hackerone.com
Connection: close
Content-Length: 123
Accept: */*
X-Auth-Token: [redacted]
Origin: https://hackerone.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.
3865.120 Safari/537.36
Sec-Fetch-Mode: cors
Content-Type: application/json
Sec-Fetch-Site: same-origin
Referer: https://hackerone.com/vairaselvamvvs
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: [redacted]

{"query":"query { node(id: \"gid://hackerone/Report/[redacted]\") { ... on Report { _id,latest_activity_at }}}
","variables":{}}
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 29 Oct 2019 17:50:48 GMT
Content-Type: application/json; charset=utf-8
Connection: close
Cache-Control: no-cache, no-store
Content-Disposition: inline; filename="response."
X-Request-Id: eb31d77a-6b54-4bcb-8007-c90f0b19307d
Set-Cookie: [redacted]
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Expect-CT: enforce, max-age=86400
Content-Security-Policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; child-src www.yout
```

Hacktivity of a private program visible to banned user if he gets invited to a program by hackbot - $500

- A user was banned from a HackerOne program
- Was able to get re-invited via hackbut (HackerOne automation)
- Unable to see anything on the web, showed Page not found
- But in the GraphQL API able to bypass the ban and see information about the program by querying it directly
- Allowed the hacker to bypass the “Page not found”



parth submitted a report to HackerOne. May 25th (3 years ago)

Summary:

The hacktivity of a private program is visible to banned user if he gets invited to a program by hackbot.

Description:

Back in 2016 i was banned by [redacted]'s private program ([redacted]) due to some conflict between me and their security team, i think they manually put me in banned users list, but few months back i was invited to [redacted]'s Program by Hackbot in the occasional invites HackerOne sends and i accepted it. But still i am not able to access their program which obviously i shouldn't as i am banned by their security team, but today i noticed in Hacktivity that i am still able to view the reports they have closed.

[redacted]

While going to [redacted] still shows me `Page not found` :

[redacted]

Also in my profile's `whitelisted_team_ids` i can see the team id of [redacted]

Also i am able to make the following requests :

1) Get Complete Hacktivity of program :

Request :

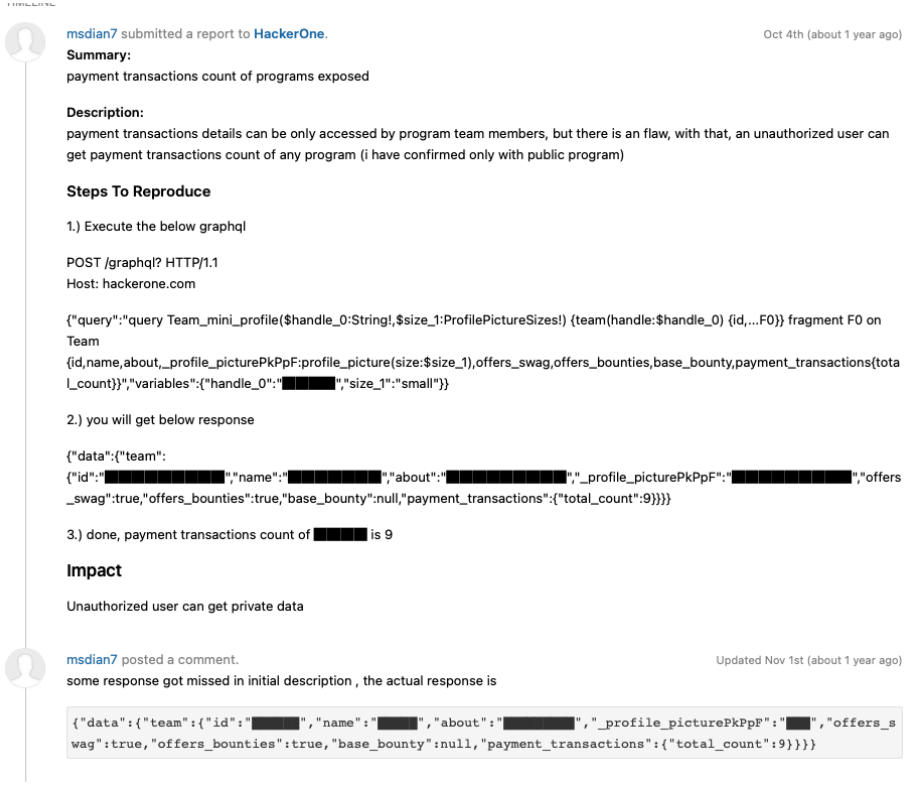
```
GET /hacktivity?sort_type=latest_disclosable_activity_at&page=1&filter=type%3Aall%20to%3A[redacted]&range
=forever HTTP/1.1
Host: hackerone.com
Connection: close
Accept: application/json, text/javascript, */*; q=0.01
X-CSRF-Token: REDACTED
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.
3359.181 Safari/537.36
Content-Type: application/json
DNT: 1
Referer: https://hackerone.com/REDACTED/hacktivity
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: REDACTED
```

Response :

```
HTTP/1.1 200 OK
Date: Fri, 25 May 2018 14:30:37 GMT
Content-Type: application/json; charset=utf-8
Connection: close
Cache-Control: private, no-cache, no-store, must-revalidate
Content-Disposition: inline; filename="response.json"
X-Request-Id: 17dafd21-a2e3-42c5-b046-e061da2a283c
Set-Cookie: REDACTED
```

Disclosure of `payment_transactions` for programs via GraphQL query - $2,500

- Very simple information disclosure
- One query didn't have any auth methods
- So would return information (payment transactions) to anyone, when it should just be returned to team members



msdian7 submitted a report to HackerOne. Oct 4th (about 1 year ago)

Summary:
payment transactions count of programs exposed

Description:
payment transactions details can be only accessed by program team members, but there is an flaw, with that, an unauthorized user can get payment transactions count of any program (i have confirmed only with public program)

Steps To Reproduce

1.) Execute the below graphql

POST /graphql? HTTP/1.1
Host: hackerone.com

{"query":"query Team_mini_profile($handle_0:String!,$size_1:ProfilePictureSizes!) {team(handle:$handle_0) {id,...F0}} fragment F0 on Team
{id,name,about,_profile_picturePkPpF:profile_picture(size:$size_1),offers_swag,offers_bounties,base_bounty,payment_transactions{total_count}}","variables":{"handle_0":"[redacted]","size_1":"small"}}

2.) you will get below response

{"data":{"team":
{"id":"[redacted]","name":"[redacted]","about":"[redacted]","_profile_picturePkPpF":"[redacted]","offers_swag":true,"offers_bounties":true,"base_bounty":null,"payment_transactions":{"total_count":9}}}}

3.) done, payment transactions count of [redacted] is 9

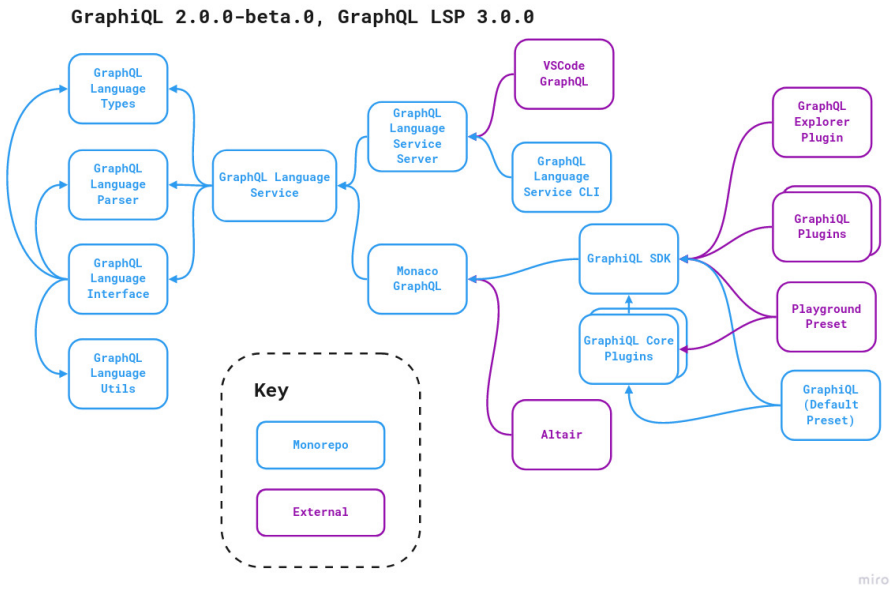
Impact

Unauthorized user can get private data

msdian7 posted a comment. Updated Nov 1st (about 1 year ago)
some response got missed in initial description , the actual response is

{"data":{"team":{"id":"[redacted]","name":"[redacted]","about":"[redacted]","_profile_picturePkPpF":"[redacted]","offers_swag":true,"offers_bounties":true,"base_bounty":null,"payment_transactions":{"total_count":9}}}}

What about GraphQL specific bugs



- Because of how GraphQL is built there are some bugs that are unique to GraphQL
- Primarily this revolves around how queries have been written in the code
- These usually require access to the source code + skills in source code review

How to hack GraphQL APIs

A simple guide

Your approach should be

- Try to introspect to find the GraphQL queries and mutations
  - If introspection is turned off, go for a traditional recon approach, push buttons, use wordlists
- Identify the business logic of each endpoint
- Craft queries to check for:
  - Information disclosure, IDORs…
- Remember the hard part of GraphQL bug hunting is the SYNTAX!
  - A great place to practice is the h101 CTF
  - In the next few weeks I'll be doing a live demo to demonstrate how this is done

Show me don't tell me

```
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
```


```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": {
        "name": "Mutation"
      },
      "types": [
        {
          "kind": "OBJECT",
          "name": "Query",
          "description": null,
          "fields": [
            {
              "name": "users",
              "description": null,
              "args": [
                {
                  "name": "id",
                  "description": null,
                  "type": {
                    "kind": "SCALAR",
                    "name": "Int",
                    "ofType": null
                  },
                  "defaultValue": null
                },
                {
                  "name": "name",
                  "description": null,
                  "type": {
                    "kind": "SCALAR",
                    "name": "String",
                    "ofType": null
                  },
                  "defaultValue": null
                },
                {
                  "name": "email",
                  "description": null,
                  "type": {
                    "kind": "SCALAR",
                    "name": "String",
                    "ofType": null
                  },
                  "defaultValue": null
                },
                {
                  "name": "email_verified_at",
                  "description": null,
```


Search Query...

No Description

FIELDS

```
users(
  id: Int
  name: String
  email: String
  email_verified_at: String
  password: String
  remember_token: String
  created_at: String
  updated_at: String
): [User]

grades(
  id: Int
  grade: Float
  comments: String
  created_at: String
  updated_at: String
): [Grade]

vulnerabilities(
  id: Int
  reporter: String
  issue: String
  information: String
  created_at: String
  updated_at: String
): [Vulnerability]

class(
  id: Int
  name: String
  description: String
  created_at: String
  updated_at: String
): [UniClass]

roles(
  id: Int
  name: String
  created_at: String
  updated_at: String
): [Role]
```

It's live demo time ☺