

Name: Sudhanshu Jain

Contact no: 7042383702

Email: sj.jain0208@gmail.com

ANAKIN ENTRANCE EXAM

Sl No	Table of Contents
1	Objective
2	<p>Task Details :</p> <ol style="list-style-type: none">1. Examine the cURL Command .<ul style="list-style-type: none">- Analyse the provided cURL command.- Identify and understand the headers, data payload, and the operation being performed.2. Understand the Persisted Query.3. Analyze the Sample URL.<ul style="list-style-type: none">- Analyzing the URL- Outcomes after analysis.4. Detailed process of generating the persisted query5. Reverse Engineering6. Conclusion
3	Links

Objective:

- Identify what kind of Request is this.
- How persisted query is generated.
- Reverse Engineer the curl command and generate the missing query.

Task Details :

1) Examine the cURL Command .

i) Analyze the provided cURL command:

Given curl command is:

[illegible]

- So, Basically this curl command is a graphql request and this is the url “ 'https://in.trip.com/flights/graphql/ctFlightGroupSearchMore' “ on which it is going .

```
curl_command = [
    "curl",
    "--location",
    "https://in.trip.com/flights/graphql/ctFlightGroupSearchMore",
    "-H", "Content-Type: application/json",
    "-d", '{"query": "query FlightSearch { flights { id flightNumber origin destination departureTime arrivalTime } }"}'
```

- The request is made through this trip.com’s GraphQL api.
- This curl command is designed to search for flights on Trip.com with specific criteria as we can see in the payload.
- This curl contains Headers and Payloads.
- Headers are specific information about:
 - what type of request it is?
 - what Response type will be accepted by the client?
 - Language preferences are specified in the header.
 - Type of content/what type of data payload?
 - Cookies
 - Security related headers are there.
- Now payload, basically the data part is the data payload in the curl command. JSON file only is called data payload. In easy words operations and variables required.

ii) Identify and understand the headers, data payload, and the operation being performed(based on given curl command):

header

- This part is header part.

```

(Sudhanshu_Jain@kali19HIS0410)-[~]
└─$ cat curl.py
import subprocess

def run_curl_command():
    try:
        # Construct the curl command
        curl_command = [
            "curl",
            "--location",
            "https://in.trip.com/flights/graphql/ctFlightGroupSearchMore",
            "--header", "accept: */*",
            "--header", "accept-language: en-GB,en;q=0.5",
            "--header", "content-type: application/json",
            "--header", "cookie: _abtest_userid=C22d7c80-686d-4856-8c16-043ff824651d; ibu_online_jump_site_result={\"site_url\":[],\"suggestion\":[]}; ibulanguage=EN; ibulocale=en_in; cookiePricesDisplayed=INR; IBU_FLIGHT_LIST_STYLE=Separate; ibu_online_home_language_match={\"isRedirect\":false,\"isShowSuggestion\":false,\"lastVisited\":true,\"region\":\"in\",\"redirectSymbol\":false}; ubtc_trip_pwa=0; _tp_search_latest_channel_name=hotels; UBT_VID=1719392331980.f483yPigQ5RE; _RF1=103.105.226.14; _RSG=dKa9nT3697GHdWfIVb.s8; _RDG=281d06b8099ed72f021dc27543b00f1a77; _RGUID=72460ddc-6a12-4527-8dbe-95f4a9e8d450; ibu_online_permission_cls_ct=1; ibu_online_permission_cls_gap=1720000440303; _combined_transactionIdK3D2e0f57c3-29ce-4103-a4d6-c0f2eda88048K26pageIdK3D10320667457N26initPageIdK3D10320667452; _bfa=1.1719392331980.f483yPigQ5RE.1.1720001620001.1720001719317.2.6.10320667457\",
            "--header", "origin: https://in.trip.com",
            "--header", "priority: u=1, i=1",
            "--header", "referer: https://in.trip.com/",
            "--header", "sec-ch-ua: \"Not(A)Brand\";v=8\", \"Chromium\";v=126\", \"Brave\";v=126\",
            "--header", "sec-ch-ua-mobile: ?0",
            "--header", "sec-ch-ua-platform: \"macOS\"",
            "--header", "sec-fetch-dest: empty",
            "--header", "sec-fetch-mode: cors",
            "--header", "sec-fetch-site: same-origin",
            "--header", "sec-gpc: 1",
            "--header", "user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36",
            "--header", "x-ibu-flt-currency: INR",
            "--header", "x-ibu-flt-language: en",
            "--header", "x-ibu-flt-locale: en-IN",

```

- Some included headers here are:

1. accept: can accept any type of request or response.
2. accept-language: Can accept British English or English as secondary language.
3. content-type: Indicates type of payload. As indicated in header content is in JSON format.
4. cookie: session-related information are stored in this.
5. origin, 'referer' : Request's Origin.
6. 'sec-*' : Security header
7. user-agent: Identifies client side software.

Data Payload

- Data Payload: This is the Standard specification is used in data payload:

```

{
    operationName: 'MyQuery',
    variables: null,
    query: `query MyQuery { id }`,
    extensions:
    {
        persistedQuery:
        {
            version: 1,
            sha256Hash: hashOfQuery
        }
    }
}

```

- This is the hierarchy of the data payload according to the JSON file written by me for the given data payload in the curl command.

Operation Name:

Variables:

request:

Head:

shoppingId:

moreCabinClass:

priceKey:

criteriaToken:

passengerCount:

tagList, extendFieldList, flagList:

abtList:

extensions:

persistedQuery:

sha256Hash

- Below is the JSON file is nothing but the data part of the curl command:

```

1  {
2    "operationName": "FlightGroupSearch",
3    "variables": {
4      "request": {
5        "head": {
6          "clientTime": "2024-07-03T15:45:22+05:30",
7          "extendFields": {
8            "BatchedId": "f7f43ca8-4c28-4688-ba7e-a825ebc38d1d"
9          }
10       },
11       "shoppingId": "AI333-BKK-DEL-20240705,UK815-DEL-BLR-20240706,5Q509-BLR-SIN-20240708,5Q706-SIN-BKK-20240709~NEWTOKEN|KLUV_QBYJQwARhbcQEBL3ABD8LoMwm8r6Iowj11j-HCNhbpUIAhoogCPf19psIogmLtrZmN7",
12       "moreCabinClass": true,
13       "priceKey": "AI333-BKK-DEL-20240705,UK815-DEL-BLR-20240706,5Q509-BLR-SIN-20240708,5Q706-SIN-BKK-20240709~NEWTOKEN|KLUV_QBYJQwARhbcQEBL3ABD8LoMwm8r6Iowj11j-HCNhbpUIAhoogCPf19psIogmLtrZmN7",
14       "criteriaToken": "tripType:RT|criteriaToken:NEWTOKEN|KLUV_QBYJQwARhbcQEBL3ABD8LoMwm8r6Iowj11j-HCNhbpUIAhoogCPf19psIogmLtrZmN7",
15       "passengerCount": 1,
16       "tagList": [
17         {
18           "key": "displayPriceType",
19           "value": "DEF"
20         }
21       ],
22       "extendFieldList": [],
23       "flagList": [],
24       "abtList": [
25         {
26           "abCode": "240319_IBU_ollrc",
27           "abVersion": "B"
28         },
29         {
30           "abCode": "240509_IBU_RFUO",
31           "abVersion": "A"
32         }
33       ]
34     },
35   },
36   "extensions": {
37     "persistedQuery": {
38       "version": 1,
39       "sha256Hash": "92b90a796ff93e8180655dda298e6208d36d027398ed7fb0b309490bcdcdbea"
40     }
41   }
42 }

```

- We can clearly see these standards specifications are there in the given payload but query part is missing.
- After extensions query data should have been there.

- JSON file attached with .zip files with name: anakin.json

2) Understand the Persisted Query

- This persisted query contains sha256 Hash.
- Lets Understand Cryptography before we dive into sha256:-
 - Before understanding sha256 lets understand Encryption and decryption.
 - Encryption converts the plaint text to cipher text, Using encryption key.
 - Decryption does the opposite, converts cipher text to the plain text using decryption key.
 - This is only called cryptography.
 - Types of cryptography
 - a) Symmetric:-
 - Symmetric used only one/same key for both encryption and decryption.
 - Example:
 - P and Q are there.
 - So before sending the data to Q, P with encrypt the data with the private key.
 - And When Q received the data he/she will require same private key to decrypt the data. Same goes for P.
 - Data can only be de-ciphered if they both have the same private key used to cipher it.
 - Private key is not send with the ciphered data it will make the data vulnerable to attacks like man in middle. Private keys here are sent through Diffie-Hellman key exchange.
 - b) Asymmetric:-
 - Asymmetric used two keys Public and private key.
 - For example P and Q.
 - P and Q both of them have their Public and Private keys.
 - Before communication P only gives its public key to Q and Q only gives its public key to P.
 - They both still have their own Private Keys protected. Private keys are not shared only public keys are shared.
 - Now P encrypt that data with Q's public key and sends data to Q.
 - Now Q uses its private key to decrypt the data as data was encrypted by Q's public key which was with P(P have public key of Q), who encrypted the data.
 - Only P's Private key can open data encrypted by P's public key. Same for Q.

- Keys are generated with the help of RSA Algorithm and then public keys are shared before the communication starts.

c) Hashing : – Hashing is basically process of generating the Hash. Hash is scrambled pieces of the actual data. Hash is nothing but a cipher text.

Focus on the persistedQuery section in the data payload:

- SHA256 Hash: SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function technique.
 - SHA-256 does not use keys for encryption. Instead, it generates a hash function.
 - For example there is a message “MyName”
 - So SHA-256 will generate a Hash value for this message.
 - This Hash value is stored and transmitted during the communication.
 - Once the communication is done the Hash value is regenerated to the message “MyName”.
 - And then the new generated hash is compared with the old stored hash value.
 - If you are exact same then data is not altered. But if they are different then the data is altered.
- **Now lets come back to the task, Persisted Query(Understand the Persisted Query):- Focus on the persistedQuery section in the data payload**
- **This persisted query contains sha256 Hash.**

```
"extensions": {  
  "persistedQuery": {  
    "version": 1,  
    "sha256Hash": "92b90a796ffd93e8180655dda298e6208d36d027398ed7fb0b309490bcdcdbea"  
  }  
}
```

"persistedQuery":

{

"version": 1,

"sha256Hash": "92b90a796ffd93e8180655dda298e6208d36d027398ed7fb0b309490bcdcdbea"

}

Investigate how the sha256Hash is generated and its role in the query process

- When a Persisted Query is sent, the browser/curl command(client) omits the query field, and instead sends an extension field with a persistedQuery object.
- And if client needs to register the hash, the query signature will be the same but include the full query text.
- **Hash is generated through the query.**
- In the given payload of the curl command in the question, query is not there but instead:
 - . Variables are given
 - . What variables are used in the query are all given
 - . Values of the variables are given in the payload.

```
"variables": {
  "request": {
    "Head": {
      "clientTime": "2024-07-03T15:45:22+05:30",
      "ExtendFields": {
        "BatchedId": "f7f43ca8-4c28-4688-ba7e-a825ebc38d1d"
      }
    },
    "shoppingId": "AI333-BKK-DEL-20240705,UK815-DEL-BLR-20240706,SQ509-BLR-SIN-20240708,SQ706-SIN-BKK-20240709^^NEWTOKEN|KLUV_QBYJQwARhhcQEBL3AbD8LoMwn8r6Iowjj11j-HCNhbpU1AHoogCPf19ps1ogmLtRzWNJHU2cxUVUldiTPiCN6",
    "moreCabinClass": true,
    "priceKey": "AI333-BKK-DEL-20240705,UK815-DEL-BLR-20240706,SQ509-BLR-SIN-20240708,SQ706-SIN-BKK-20240709^^NEWTOKEN|KLUV_QBYJQwARhhcQEBL3AbD8LoMwn8r6Iowjj11j-HCNhbpU1AHoogCPf19ps1ogmLtRzWNJHU2cxUVUldiTPiCN6",
    "criteriaToken": "tripType:RT|criteriaToken:NEWTOKEN|KLUV_QBYJQwARhhcQEBL3AbD8LoMwn8r6Iowjj11j-HCNhbpU1AHoogCPf19ps1ogmLtRzWNJHU2cxUVUldiTPiCN6jrr4XTqkRuGVSAEkASgD7sikBjq2c0NeZ0CsTVPY-Xgk1lk3QW52UKtiybrsAVI"
  }
}
```

- But query it self is missing in the given curl command's payload, through which the hash is generated.
- Query is basically to fetch user details
- Instead of sending full query, we can persist it on the server and generate a hash for it, which only is called Persisted Query.
- So query works like this (basic mechanism):-
 - Client/browser: Sends the hash of the query instead of the full query.
 - Server: Looks up the query corresponding to the hash, and generates the result.
 - query document is taken and hash is returned.
- So we need to reverse Engineer the query, which is the main task.
- We need to construct Query part.

- We will do more investigation on this **Persisted Query after** Analysing the given sample URL.

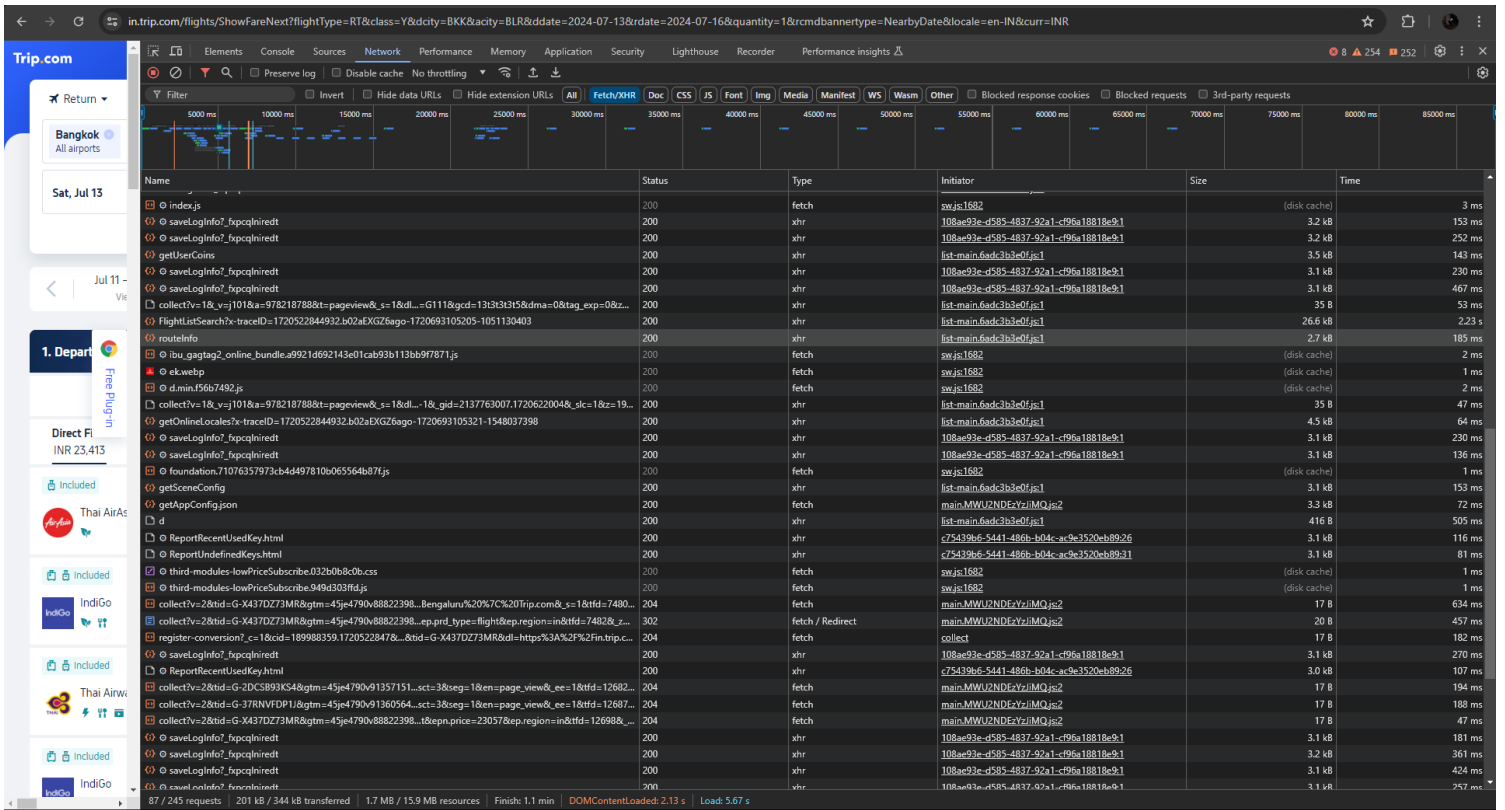
Analyse the Sample URL -

- Use the sample URL to simulate a flight search on Trip.com:
- [Booking Link](#)

3) Analyze the Sample URL.

- Use the sample URL to simulate a flight search on Trip.com:
 - [Booking Link](#)
-

- Examining the given URL using browser developer tools.
- If we do fn+f12 we will open browser developer tool.
- There we can go to Network > Fetch/XHR.
- There we can see all the calls and traffic.



- There we go to route info for different requests then we will see something similar.
- As explained earlier in the Payload part we can see the same
 - If we go to the payload part we will see the same Standard specification used in data payload:

```

{
  operationName: 'MyQuery',
  variables: null,
  extensions:
  {
    persistedQuery:
    {
      version: 1,
      sha256Hash: hashOfQuery
    }
  }
}

```

in.trip.com/flights/ShowFareNext?flightType=RT&class=Y&dcity=BKK&acity=BLR&ddate=2024-07-13&rdate=2024-07-16&quantity=1&rcmdbannertype=NearbyDate&locale=en-IN&curr=INR

Trip.com

Return

Bangkok All airports

Sat, Jul 13

1. Depart

Free Plug-in

Direct First INR 23,413

Included Thai Air

Included IndiGo

Included Thai Airw

Included IndiGo

97 / 256 requests | 235 kB / 379 kB transferred | 1.7 MB

Network

Fetch/XHR

Request Payload

```
{
  "operationName": "routeInfo",
  "variables": {},
  "extensions": {}
}
{
  "persistedQuery": {
    "version": 1,
    "sha256Hash": "058f6f851af758ecbc103b8aa4cb798c5cd597e3ab4874065ba9bc4df11f500"
  },
  "operationName": "routeInfo",
  "variables": {},
  "request": {
    "departure": "BKK",
    "arrival": "BLR",
    "searchCriteria": {
      "passengerCount": {
        "adult": 1,
        "child": 0,
        "infant": 0
      }
    }
  },
  "passengerCount": {
    "adult": 1,
    "child": 0,
    "infant": 0
  },
  "searchSegmentList": [
    {
      "departureCity": "BKK",
      "arriveCity": "BLR",
      "departureCity": "BLR",
      "arriveCity": "BKK"
    }
  ],
  "tripType": "RT"
}
```

in.trip.com/flights/bangkok-to-bengaluru/tickets-bkk-blr?dcity=bkk&acity=blr&ddate=2024-07-13&rdate=2024-07-16&triptype=rt&class=y&lowpricesource=searchform&quantity=1&searchboxarg=t&nonstoponly=off&locale=en-IN&curr=INR

Trip.com

Return 1 Adult

Bangkok All airports

Sat, Jul 13

1. Departing to Bengaluru

Free Plug-in

Direct First INR 23,413

Low INR 23,05

Included Thai AirAsia 20:3 DMK

Included IndiGo 12:2 BKK

Included Thai Airways 21:2 BKK

Included IndiGo 10:2

50 / 79 requests | 155 kB / 179 kB transferred | 1.6 MB

Network

Fetch/XHR

Request Payload

```
{
  "operationName": "routeInfo",
  "variables": {},
  "extensions": {}
}
{
  "persistedQuery": {
    "version": 1,
    "sha256Hash": "058f6f851af758ecbc103b8aa4cb798c5cd597e3ab4874065ba9bc4df11f500"
  },
  "operationName": "routeInfo",
  "variables": {},
  "request": {
    "departure": "BKK",
    "arrival": "BLR",
    "searchCriteria": {
      "passengerCount": {
        "adult": 1,
        "child": 0,
        "infant": 0
      }
    }
  },
  "passengerCount": {
    "adult": 1,
    "child": 0,
    "infant": 0
  },
  "searchSegmentList": [
    {
      "departureCity": "BKK",
      "arriveCity": "BLR",
      "departureCity": "BLR",
      "arriveCity": "BKK"
    }
  ],
  "tripType": "RT"
}
```

The screenshot shows a flight search interface on [in.trip.com](https://in.trip.com/flights/bangkok-to-bengaluru/tickets-bkk-blr?city=bkk&city=blr&date=2024-07-13&date=2024-07-13&tripType=rt&class=y&lowPriceSource=searchform&quantity=1&searchboxarg=t&nonstoponly=off&locale=en-IN&curr=INR). The search results show flights from Bangkok to Bengaluru for Saturday, July 13th. The developer tools are open, showing a list of network requests. The 'routeInfo' request is selected, and its payload is visible in the 'Request Payload' tab. The payload is a JSON object containing flight details and search criteria.

```

{
  "operationName": "routeInfo",
  "variables": {
    "departure": "BKK",
    "arrival": "BLR",
    "searchCriteria": {
      "passengerCount": {
        "adult": 1,
        "child": 0,
        "infant": 0
      }
    }
  },
  "extensions": {
    "persistedQuery": {
      "version": 1,
      "sha256Hash": "058f6f851af758e6bc103b8aa4cb798c5cd597e3ab4874065ba9bc4df11f500"
    },
    "operationName": "routeInfo"
  }
}

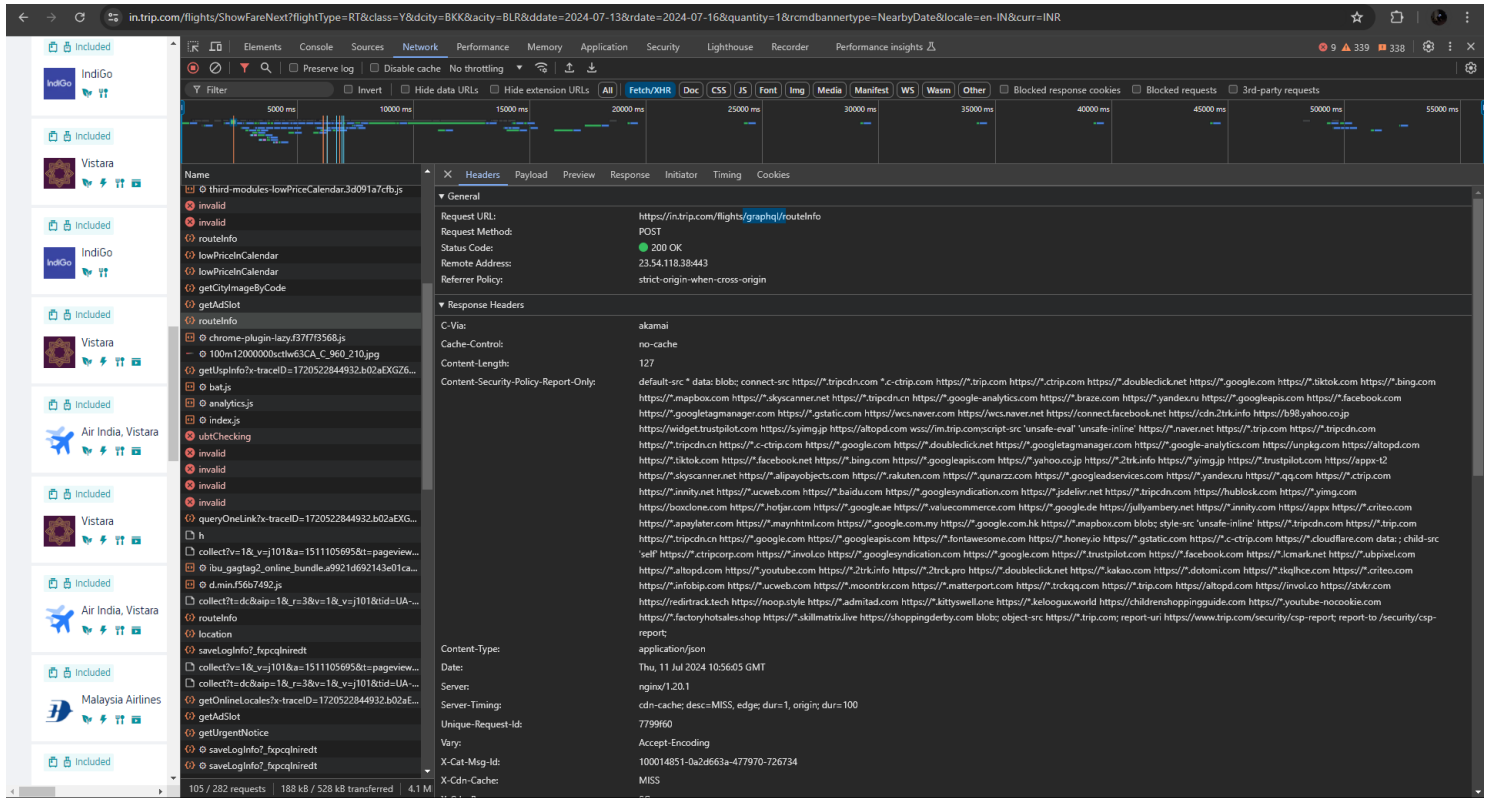
```

• Outcomes after analysis(Found 2 important information):

- **First Important is Query parameters are missing:** We can see same Standard specification is followed but here also query is missing.
- The URL payload is also persisted query so query field is missing over here as well.
- Observed many requests but query is missing in all of them.
- If would have found query somewhere, so we could have remade the query but no query present.
- **Second important info we found is that provided URL is a page load URL.**
- **To load that page multiple AJAX request are sent same of which are graphql request.**

Request URL:	https://in.trip.com/flights/graphql/routeInfo
Request Method:	POST
Status Code:	200 OK

- As we can see this is also GraphQL request.
- this one thing is clear the query is of graphql for sure as we can see in payload of the trip.com URL is graphql only.



- So our missing query is graphql for sure.

4) Detailed process of persisted query is generated and Reverse Engineering:

- Before diving into this lets run our curl command.
- So lets run the provided curl command first.

- ```

_jain@kali19WIS0418) ~/Desktop
execute_Curl.py
ceded!

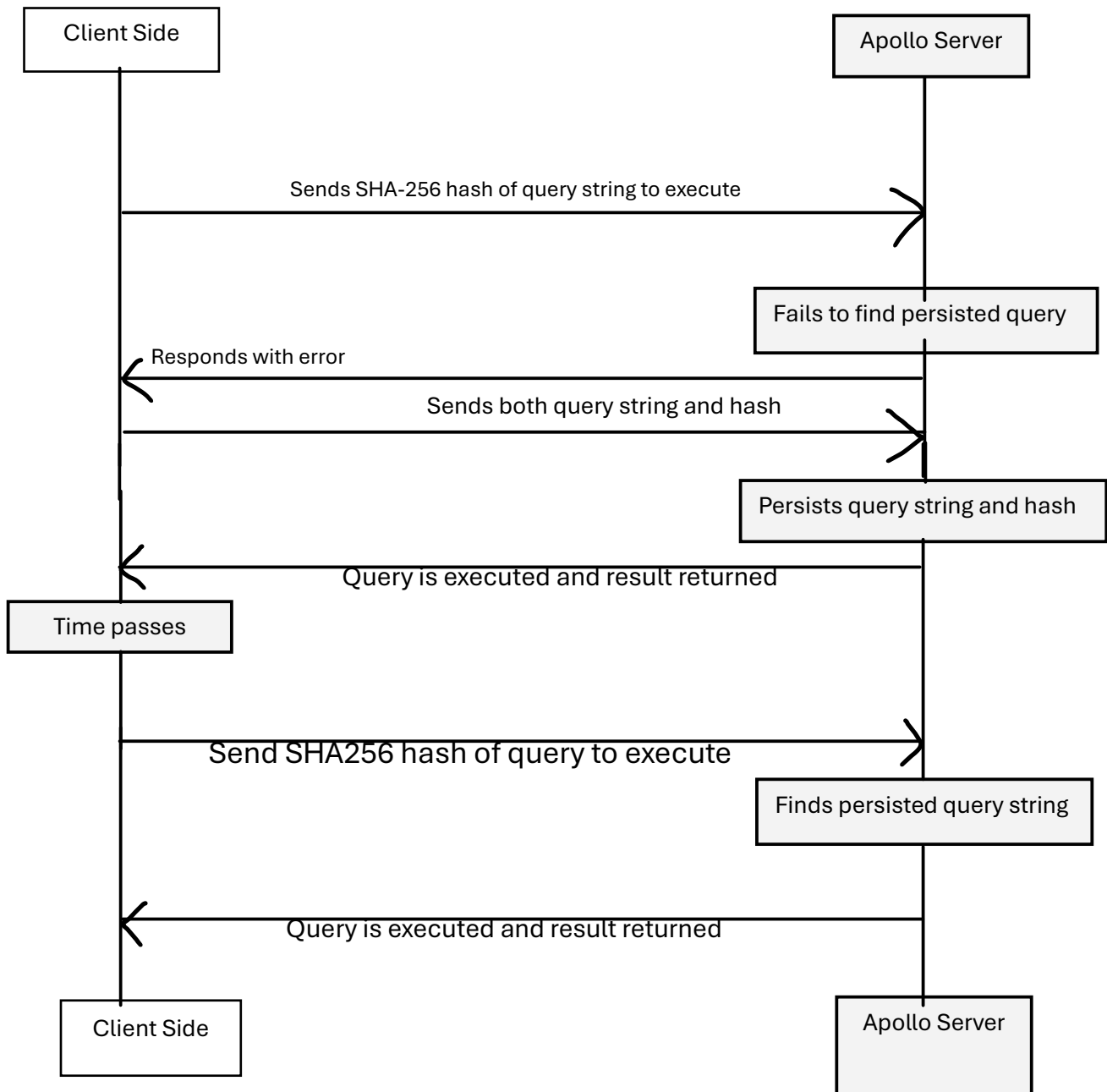
{"message": "PersistedQueryNotFound", "extensions": {"code": "PERSISTED_QUERY_NOT_FOUND", "exception": {"stacktrace": ["PersistedQueryNotFoundError: PersistedQueryNotFound", " at Object.<anonymous> (/opt/nodeapp/node_modules/apollo-server-core/dist/requestPipeline.js:58)", " at Generator.next (<anonymous>)", " at runMicrotasks (<anonymous>)", " at processTicksAndRejections (node:internal/process/task_queues:96:5)"]}}}

```

- We got an error persisted query not found
- We can see we got something related to apollo-server-core
- We got something related to apollo server so from here we will further research about apollo servers and clients on there website and how is persisted query generated here and try to reverse Engineer it.

## **Detailed process of how persisted query is generated**

- So from the above error found we got to know this is related to apollo servers and apollo client.
- So what happens, Queries are sent by client to Apollo server in the form of HTTP request that include GraphQL String .
- But the size of the query string might be large depending on the schema.
- Large query size leads to increased latency and network usage which can affect the client performance.
- So, to improve network performance for large query strings, Apollo Server generates Automatic Persisted Queries (APQ)  
\_\_\_\_\_.
- A persisted query is a query string that's cached on the server side which a unique SHA256-Hash.
- So in order to reduce latency and network usage client sends this persisted query sha256-hash instead of sending the large query string.
- Which in turns reduces the size drastically and does not degrades performance.
- Below is working mechanism or diagram or how persisted query is generated:





- **Explanation:**

- There are servers called CDN(Content delivery network).
- There are many CDNs for fast processing.
- Caching of full graphQL queries is done here.
- So workflow or we can say path on which how it is working is:

- **Happy Path:-**

- Client sends query signature with no query field
- Server looks up query based on hash, if found, it resolves the data
- Client receives data and completes request

- **Missing hash path:-**

- Client sends query signature with no query field
- Server looks up query based on hash, none is found
- Server responds with NotFound error response
- Client sends both hash and query string to Server
- Server fulfills response and saves query string + hash for future lookup
- Client receives data and completes request

## 5) **Reverse Engineering the query**

- Run the curl command: So when we ran the curl command most of the time error message is given in response that persisted query not found.

- When the query signature is received by a backend, if it is unable to find the hash previously stored or unable to cache it, it must send back response error” persistedQueryNotFound”.
- Which clearly means it is not cached on the CDN server.
- This is only Missing hash path

```
(Sudhanshu_Jain@kali19MISO418)-[~/Desktop]
$ python execute_Curl.py
Command succeeded!
Response:
{"errors":[{"message":"PersistedQueryNotFound","extensions":{"code":"PERSISTED_QUERY_NOT_FOUND","exception":{"stacktrace":["PersistedQueryNotFoundError: PersistedQueryNotFound"," at Object.<anonymous> (/opt/nodeapp/node_modules/apollo-server-core/.js:67:52)"," at Generator.next (<anonymous>)"," at fulfilled (/opt/nodeapp/node_modules/apollo-server-core/dist/requestPipeline.js:5:58)"," at runMicrotasks (<anonymous>)"," at processTicksAndRejections (node:internal/process/task_queues
```

- Now sometime, When I ran the command again and again. Instead of error we got a response:

```
(Sudhanshu_Jain@kali19MISO418)-[~/Desktop]
$ python execute_Curl.py
Command succeeded!
Response:
{"data":{"flightGroupSearch":{"resultBasic":{"criteriaToken":"","currency":"","regionRoute":"","flightClass":"","hasOtherPolicy":false,"__typename":"ResultBasicPayload"},"flightProductList":[],"business":{"creditCard":{"discount":null,"paymentList":null,"__typename":"CreditCardPayload"},"__typename":"FlightGroupBusinessPayload"},"recommendList":null,"bookingFeeTCUrl":null,"bookingFeeHiddenDesc":null,"abTestingList":null,"__typename":"FlightGroupSearchResponsePayload"}}
```

```

1 {
2 "data": {
3 "flightGroupSearch": {
4 "resultBasic": {
5 "criteriaToken": "",
6 "currency": "",
7 "regionRoute": "",
8 "flightClass": "",
9 "hasOtherPolicy": false,
10 "__typename": "ResultBasicPayload"
11 },
12 "flightProductList": [],
13 "business": {
14 "creditCard": {
15 "discount": null,
16 "paymentList": null,
17 "__typename": "CreditCardPayload"
18 },
19 "__typename": "FlightGroupBusinessPayload"
20 },
21 "recommendList": null,
22 "bookingFeeTCUrl": null,
23 "bookingFeeHiddenDesc": null,
24 "abTestingList": null,
25 "__typename": "FlightGroupSearchResponsePayload"
26 }
27 }
28 }
29
```

- This is the data we got in response without error
- This is a Happy path.

- Now we will try to deduce the Query from it and will find its hash.
- *After reverse Engineering and researching the query, I was able to generate is:*

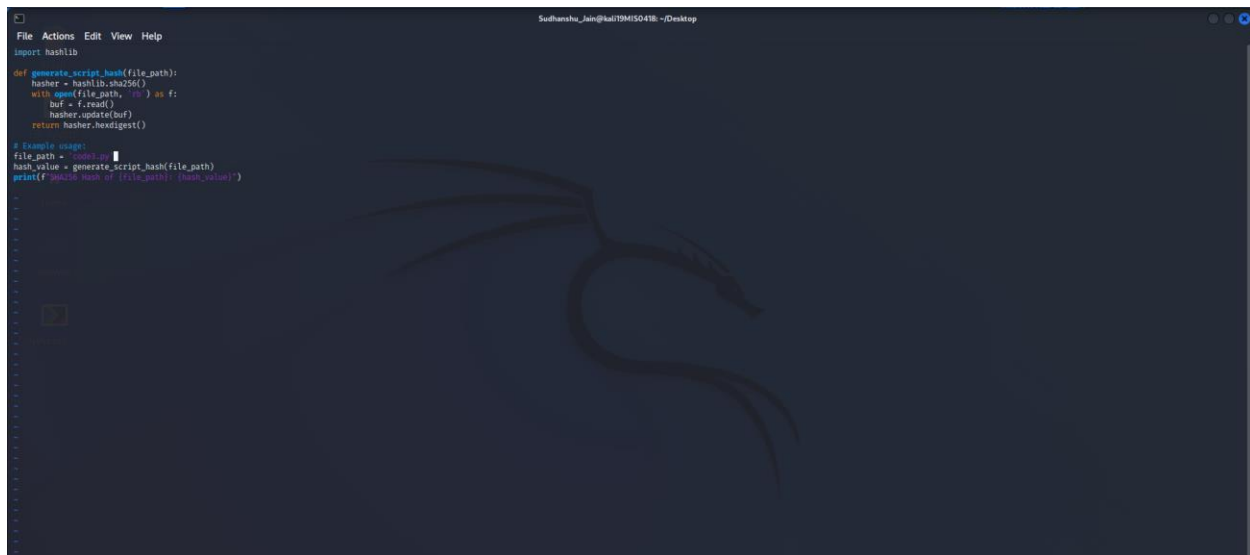
```
1 query flightGroupSearch {
2 flightGroupSearch {
3 resultBasic {
4 criteriaToken
5 currency
6 regionRoute
7 flightClass
8 hasOtherPolicy
9 __typename
10 }
11 flightProductList {
12 }
13 business {
14 creditCard {
15 discount
16 paymentList
17 __typename
18 }
19 __typename
20 }
21 recommendList {
22 }
23 bookingFeeTCUrl
24 bookingFeeHidenDesc
25 abTestingList {
26 }
27 __typename
28 }
29 }
30
```

```
query flightGroupSearch {
 flightGroupSearch {
 resultBasic {
 criteriaToken
```

```
 currency
 regionRoute
 flightClass
 hasOtherPolicy
 __typename
 }
 flightProductList {
 }
 business {
 creditCard {
 discount
 paymentList
 __typename
 }
 __typename
 }
 recommendList {

 }
 bookingFeeTCUrl
 bookingFeeHidenDesc
 abTestingList {
 }
 __typename
}
}
```

Hash generated for this query using the python script I wrote is:



```
File Actions Edit View Help
Sudhanshu_Jain@kali19MIS0418: ~/Desktop
import hashlib

def generate_script_hash(file_path):
 hasher = hashlib.sha256()
 with open(file_path, 'r') as f:
 buf = f.read()
 hasher.update(buf)
 return hasher.hexdigest()

Example usage:
file_path = 'code3.py'
hash_value = generate_script_hash(file_path)
print(f'SHA256 Hash of {file_path}: {hash_value}')
```

Hash: 246250c4b6731b9bdb73531e2d3b4cc00d5e049ea21a95eae7a2fea7b3631e1a

```
(Sudhanshu_Jain@kali19MIS0418) - [~/Desktop]
$ python code_hasher.py
SHA256 Hash of code3.py: 246250c4b6731b9bdb73531e2d3b4cc00d5e049ea21a95eae7a2fea7b3631e1a
```

The Python script to Generate the hash is provided with .zip file name as: code\_hash\_generate.py

## 6) Conclusion:

- In this project I was basically supposed to reverse engineer and deduce the missing query. Also, was supposed to match the Hash of the deduced query with the one provided in question.
- It was a graphql request.
- As a result the Hash of the deduced query is different from the provided hash.
- It could be due to following reasons why hash is not matching:
  - There can be some more or less space in the actual query.
  - Object names could be different in the actual query.
  - Could be sensitive to spaces.
  - Could be sensitive to object names.
- At least it provides the high level idea how the query is going to look like.

## 7)Links:

### Important Links which helped with throughout my research:

- <https://www.apollographql.com/docs/router/executing-operations/requests/>
- For deducing the query: <https://graphql.org/learn/queries/>
- <https://graphql.com/learn/arguments/>
- <https://graphql.com/learn/arguments/>
- <https://jsonformatter.org/>

### **Google Drive links:**

- Python Script to generate hash: code\_hash\_generate.py
- Python script to run the curl command: execute\_Curl\_command.py
- JSON FILE: anakin.json

