Name: Sudhanshu Jain

Contact no: 7042383702

Email: sj.jain0208@gmail.com

|  |  |
| --- | --- |
| Serial No | Table of Contents |
| 1 | Objective |
| 2 | Task Details:   1. Examine the cURL Command  * Analyse the provided cURL command. * Identify and understand the headers, data payload, and the operation being performed.  1. Understand the Persisted Query 2. Analyze the Sample URL  * Analyzing the URL * Outcomes after analysis  1. Detailed process of generating the persisted query 2. Reverse Engineering 3. Conclusion |
| 3 | References |

**Objective:**

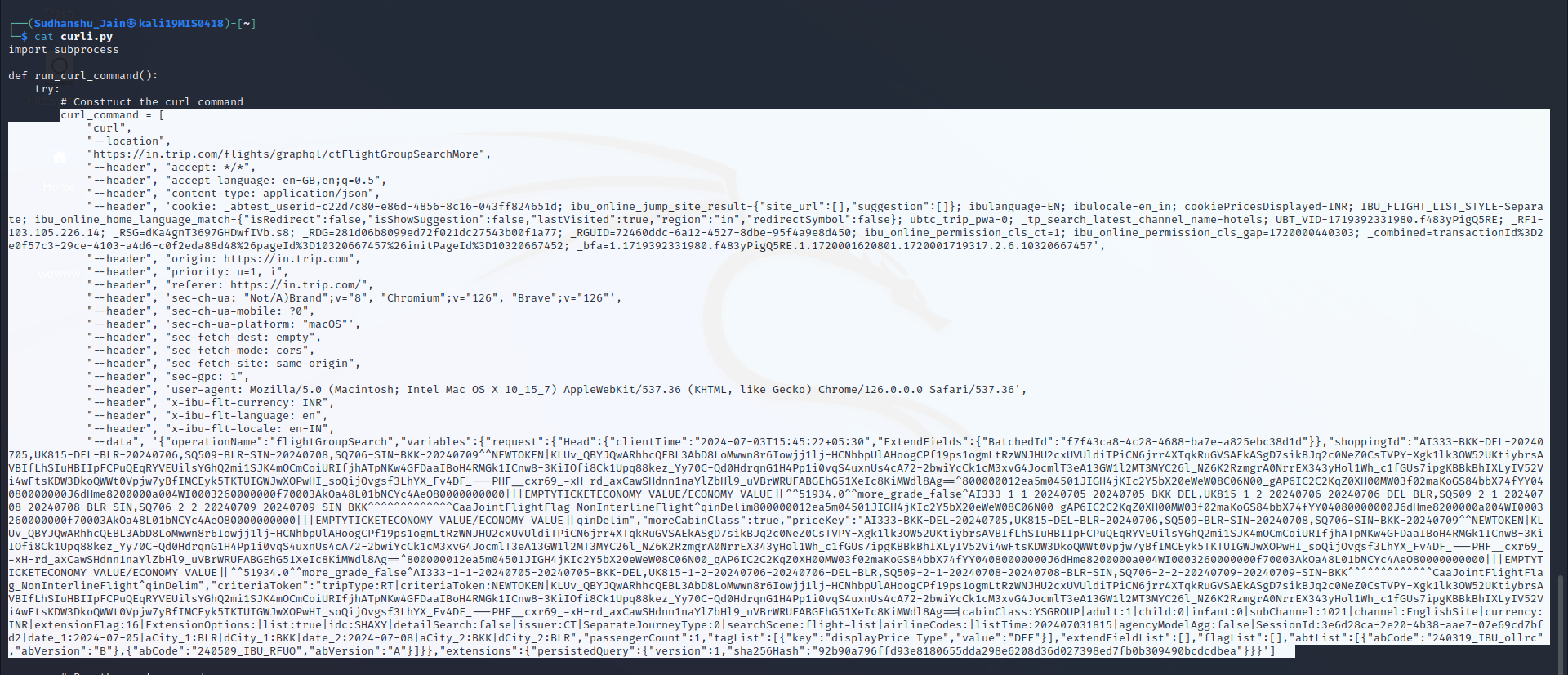
* Analyze the given curl command
* Identify how persisted query is generated
* Reverse Engineer to generate the missing query

**Task Details:**

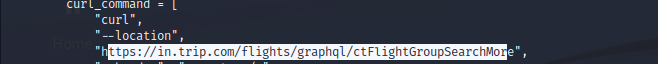
**1) Examine the cURL Command**

* + 1. ***Analyze the provided cURL command:***

Given curl command is:



* So, this curl command is a graphql request and this is the url ***'https://in.trip.com/flights/graphql/ctFlightGroupSearchMore'***



* The request is made through 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 command 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

* In the payload, the data part is the “data payload” in the curl command. The data payload is in the JSON format. It has different attributes like operationName, variables, extensions etc.
  + 1. ***Identify and understand the headers, data payload, and the operation being performed (based on given curl command):***

***header***

* This is the header part in the curl Command



* Some included headers 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 considered and is in JSON format:

{

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.

operationName:

variables:

request:

Head:

shoppingId:

moreCabinClass:

priceKey:

criteriaToken:

passengerCount:

tagList

extendFieldList:

flagList:

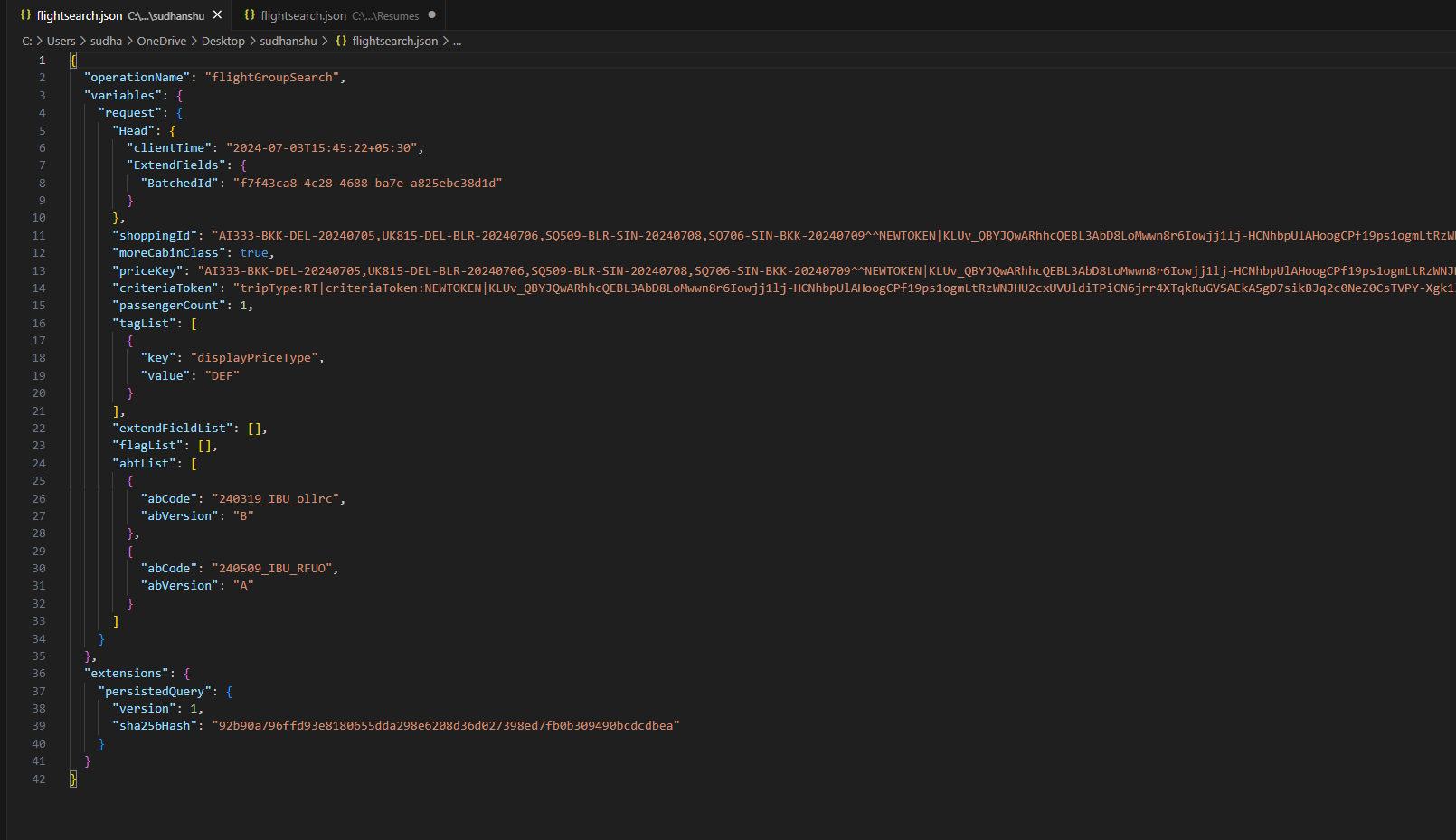
abtList:

extensions:

persistedQuery:

sha256Hash

* Below is the JSON file which is nothing but the data part of the curl command (formatted using online json formatter):



* We can clearly see these standards specifications are there in the given payload but query part is missing.
* You will find JSON file link below in the conclusion part.

1. **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**
* **Symmetric**:-
  + - Symmetric used only one/same key for both encryption and decryption.
    - For Example:
      * Say P and Q are communicating with each other.
      * So before sending the data to Q, P will 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 deciphered if they both have the same private key used to cipher it.
      * Private key is not sent with the encrypted data, it will make the data vulnerable to attacks like man-in-the-middle. Private keys here are sent through Diffie-Hellman key exchange.
* **Asymmetric:-**
  + Asymmetric uses two keys Public and private key.
  + For example,
    - P and Q are communicating with each other.
    - 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.
* **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 they 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.**

**A black background with numbers

Description automatically generated**

"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 includes 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
  + Values of the variables are given in the payload

A black background with orange and black text

Description automatically generated

* But query itself is missing in the given curl command’s payload, through which the hash is generated.
* Query is basically to fetch the flight details
* Instead of sending full query, we can persist it on the server and generate a hash for it, which 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.
* Client generally generates the hash using the query. More details around this are explained towards the last section of this document.
* 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.

A white background with black text

Description automatically generated

1. **Analyze the Sample URL.**

A white background with black text

Description automatically generated

* 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.
* A screenshot of a computer

  Description automatically generated
* Multiple requests to server are generated while opening the given url on the browser.
* There if we go to routeinfo request then we will see similar graphql request (similar to curl command) and data payload in the request looks in similar Json format.
* As explained earlier in the Payload part we can see the same JSON Payload structure
* 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

}

}

}

A screenshot of a computer program

Description automatically generated

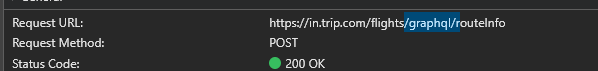
A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

* **Outcomes after analysis (Found 2 important information):**
* First Important is Query parameter is 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.
* Second important info we found is that provided URL is a page load URL .
* To load that page multiple AJAX requests are sent to the server and few of the requests are graphQL requests with similar payload structure.



* As we can see this is also GraphQl request.
* One thing is clear the query is of graphQl for sure as we can see in payload of the trip.com’s curl command also has graphql http request.

**A computer screen shot of a program

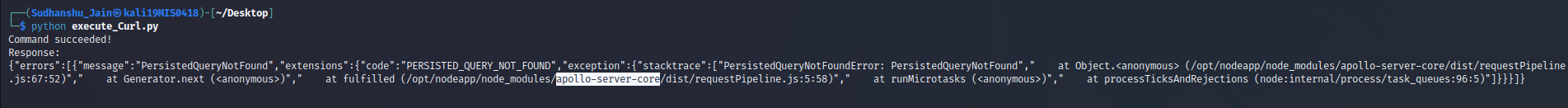
Description automatically generated**

* So our missing part is graphQL query which is hashed and persisted on the server for performance improvements.

1. 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.
* The Python script to run the curl command provided with .zip file name as: ***execute\_Curl\_command.py***
* I will be using a python script written by me to run the curl command in Kali Linux. A screen shot of a computer screen

  Description automatically generated
* So, When I ran the curl command provided then I got the response like this:



* 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 their 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 (referenced from apollo graphql website):

Client Side­

Apollo Server

Sends SHA-256 hash of query string to execute



Fails to find persisted query

Responds with error



Sends both query string and hash



Persists query string and hash

Query is executed and result returned



Time passes



Send SHA256 hash of query to execute



Finds persisted query string



Query is executed and result returned



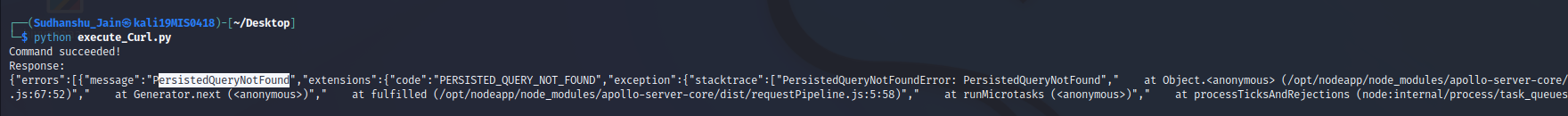
Client Side

Apollo Server

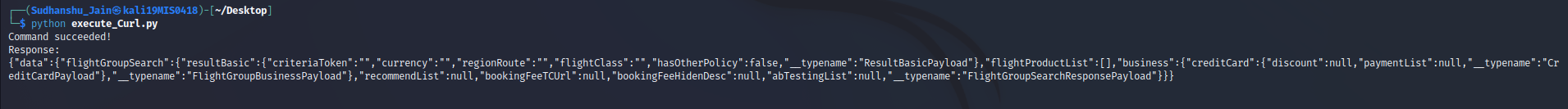
* **Explanation:**
* There are servers called CDN(Content delivery network).
* There are many CDNs for fast processing.
* Caching of full grapHQL queries is done here.
* ­So workflow or we can say path on which how it is working is:
* **Happy Path:-**
* Client sends query signature with no query field
* 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

1. **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



* Now sometimes, when I ran the command again and again. Instead of error we got a response:

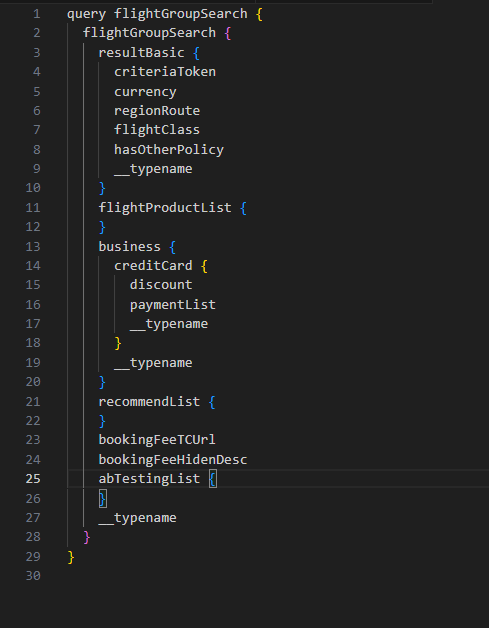


Here is the formatted version of the above response in JSON format.

A computer screen shot of text

Description automatically generated

* 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 the below query
* It is a GraphQL query. Took help of graphql tutorial online to define this query
* Above curl command response attributes helped in defining this query
* Also, variables used in the curl command’s data payload helped in defining the arguments in the graphql query



Find same query in the attached zip file with the name **Reverse\_engineered\_graphql\_query.txt**

Here is the Python Script for generating the hash for the above query.

A black dragon on a blue background

Description automatically generated

**Hash:** 246250c4b6731b9bdb73531e2d3b4cc00d5e049ea21a95eae7a2fea7b3631e1a



1. 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.

References:

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/>