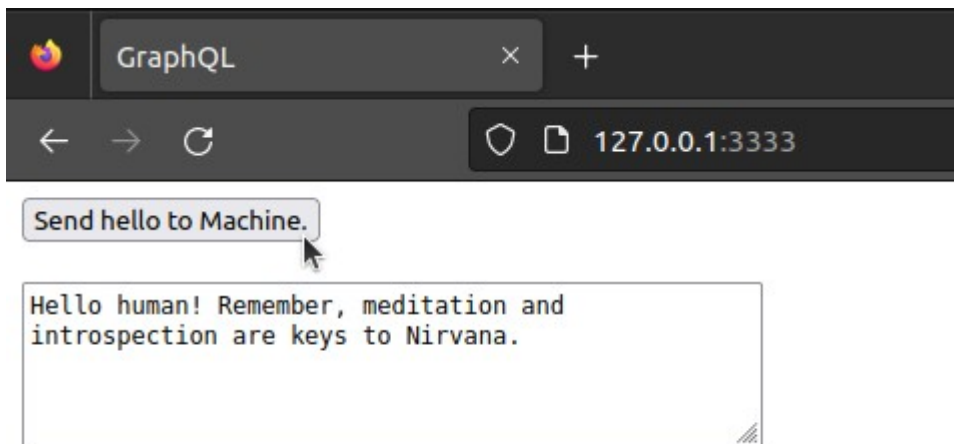


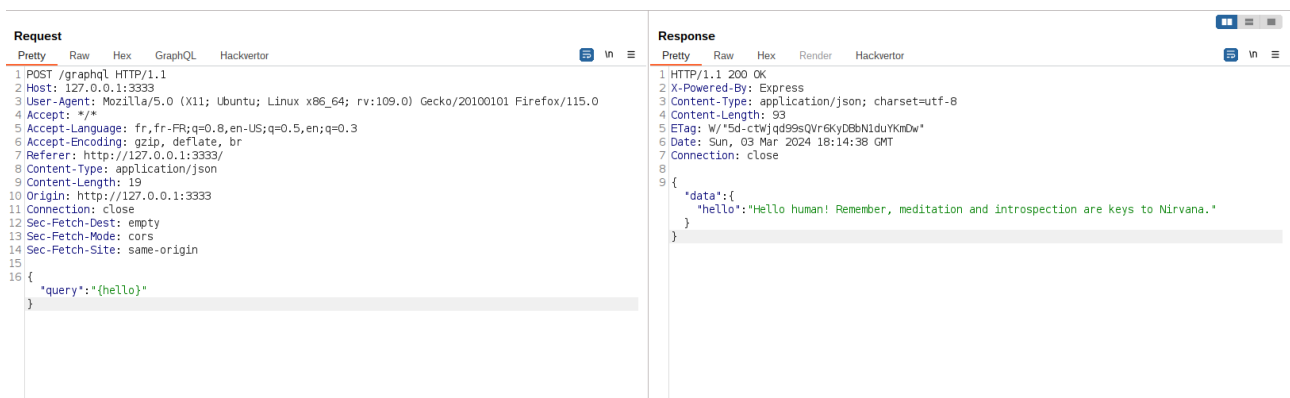
# API Part 1

## #1 Repérage

La page principale du challenge propose de dire bonjour à la machine. En cliquant sur la bouton, nous remarquons un message s'afficher dans la zone de texte.



Si l'on intercepte la requête en utilisant BurpSuite, nous voyons une requête **POST** vers `/graphql`.



Le message de la machine contient le mot clef « **Introspection** » qu'il fallait réussir à relier à la techno de l'API qu'est **GraphQL**. Puis se renseigner sur le principe et ce que permet l'introspection d'une API GraphQL.

Ressources utiles :

- [GraphQL.org](https://graphql.org)
- [Hacktricks – GraphQL](https://hacktricks.com/graphql)
- [PortSwigger – Use BurpSuite for GraphQL APIs](https://portswigger.com/web-security/graphql)
- [YesWeHack - Exploiting GraphQL APIs](https://yeswehack.com/articles/exploiting-graphql-apis/)

## #2 Exploitation

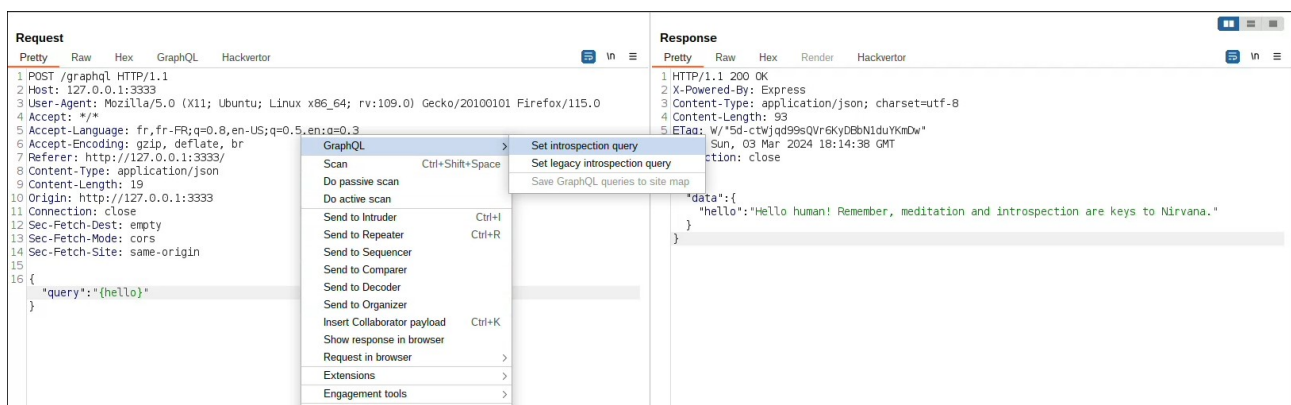
En fonction de vos recherches, plusieurs options s'offrent à vous. La première étant d'utiliser les fonctionnalités de BurpSuite liées à GraphQL. L'autre étant de copier une requête d'introspection et de la formater correctement dans votre requête.

Dans le cas d'une requête d'introspection sur plusieurs lignes, qui n'est pas pris en charge par l'argument *query* vous auriez pu utiliser la commande : **echo 'payload' | tr -d '\n'** avec payload étant la requête d'introspection.

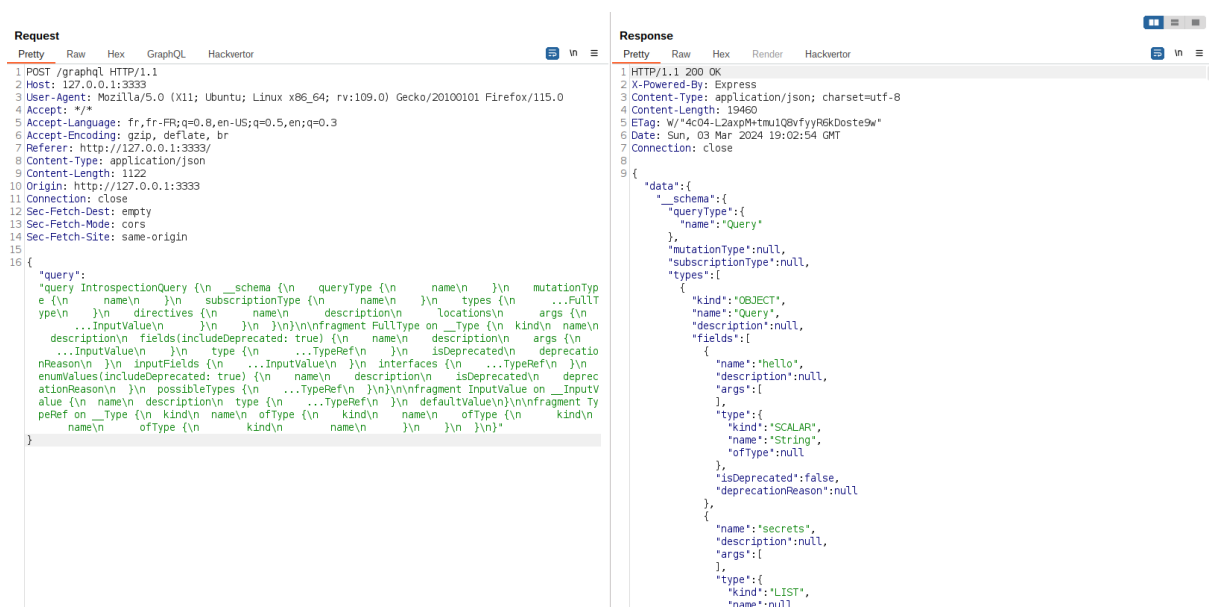
### ## Exemple 1

La [ressource suivante](#) présente comment utiliser les nouvelles fonctionnalités de BurpSuite concernant les APIs GraphQL (fonctionnalités dispo dans la version gratuite Community).

Sur la requête vers */graphql*, un clique droit permet d'afficher le menu contextuel en rapport avec GraphQL. Il suffisait de sélectionner « Set Introspection Query » afin d'obtenir la requête d'introspection.



Une fois envoyée, vous obtenez le schéma complet utilisé par l'API.



## ## Exemple 2

La [ressource suivante](#) présente une requête facilement utilisable. Elle est donnée en une seule ligne et est compactée.

So, if introspection is authorized on your target it's a good news, you will have the possibility to see all useful information to inspect and go deeper on it.

## How to perform introspection in GraphQL ?

This is the full request to perform you GraphQL introspection on your target (if enabled):

```
schema { queryType(name) mutationType(name) subscriptionType(name) types { ...FullType } directives { name description
```

The server should respond with the full schema (query, mutation, objects, fields...). Even if schema is displayed in JSON, it can be quickly unreadable. In my opinion, once you have the schema, the best way is to import it in a tool like **"GraphQL Voyager"** (I talk about below in "Tools" chapter).



Une fois collée dans BurpSuite vous obtenez le schéma complet utilisé par l'API.

Request	Response
<pre> Pretty Raw Hex GraphQL Hacktor 1 POST /graphql HTTP/1.1 2 Host: 127.0.0.1:3333 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: */* 5 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://127.0.0.1:3333/ 8 Content-Type: application/json 9 Content-Length: 721 10 Origin: http://127.0.0.1:3333 11 Connection: close 12 Sec-Fetch-Dest: empty 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Site: same-origin 15 16 {   "query":     /*( schema(queryType(name)mimeType(subscriptionType(name)types(...FullType)directives       (name description locations args(...InputValue)))fragment FullType on _Type(kind name descri       tion fields(includeDeprecated:true)(name description args(...InputValue))type(...TypeRef)isDep       recated deprecationReason(inputFields(...InputValue)interfaces(...TypeRef)enumValues(includeDe       precated:true)(name description isDeprecated deprecationReason)possibleTypes(...TypeRef))fragm       ent 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))))))))*/ } </pre>	<pre> Pretty Raw Hex Render Hacktor 1 HTTP/1.1 200 OK 2 X-Powered-By: Express 3 Content-Type: application/json; charset=utf-8 4 Content-Length: 19530 5 ETag: W/"4c4a-sm7EBLqOCAPdgDXqkbeME5+4ic" 6 Date: Sun, 03 Mar 2024 18:38:27 GMT 7 Connection: close 8 9 {   "data":{     "_schema":{       "queryType":{         "name":"Query"       },       "mutationType":null,       "subscriptionType":null,       "types":[         {           "kind":"OBJECT",           "name":"Query",           "description":null,           "fields":{             {               "name":"hello",               "description":null,               "args":{                 ],               "type":{                 "kind":"SCALAR",                 "name":"String",                 "ofType":null               },               "isDeprecated":false,               "deprecationReason":null             },             {               "name":"secrets",               "description":null,               "args":{                 ],               "type":{                 "kind":"LIST",                 "ofType":null               }             }           ]         }       ]     }   } } </pre>

### #3 Flag

Pour obtenir le flag, il fallait s'intéresser au schéma retourné par l'API et surtout aux parties qui étaient spécifique aux données gérée par l'instance GraphQL.

La méthode la plus simple était d'effectuer une recherche par mot clef en utilisant la fonctionnalité de BurpSuite. On cherche « Flag » et tada !

[illegible]