玩转graphQL

原创 先锋情报站 酒仙桥六号部队 2020-11-30

这是 **酒仙桥六号部队** 的第 **118** 篇文章。
全文共计4257个字，预计阅读时长12分钟。

前言

在测试中我发现了很多网站开始使用GraphQL技术，并且在测试中发现了其使用过程中存在的问题，那么，到底GraphQL是什么呢？了解了GraphQL后能帮助我们在渗透测试中发现哪些问题呢？

在测试中，我们最常见的graphql的数据包就像图中一样：

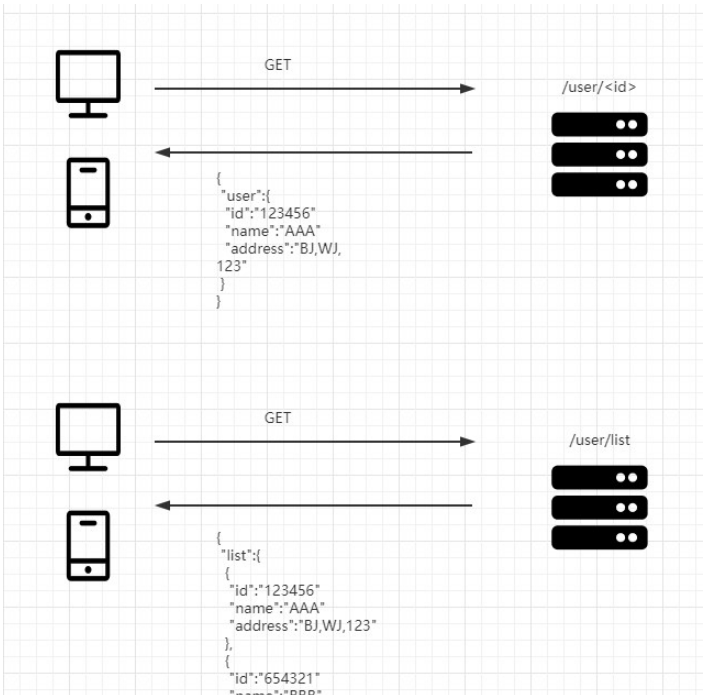
和json类似的格式，但其中包含了很多换行符\n，当你遇到这种结构的请求时，请多留心测试一下GraphQL是否安全。

前置知识

什么是GraphQL

GraphQL 是一个用于 API的查询语言，使用基于类型系统来执行查询的服务（类型系统由你的数据定义）。GraphQL 并没有和任何特定数据库或者存储引擎绑定，而是依靠你现有的代码和数据支撑。

如果你了解REST API会更快地了解它。像REST API，往往我们的请求需要多个API，每个API是一个类型。比如：http://www.test.com/users/{id} 这个API可以获取用户的信息；再比如：http://www.test.com/users/list 这个API可以获取所有用户的信息。



```
  name : BBB
  "address":"....."
 }
}
}
```

在graphql中则不需要这么多api来实现不同的功能，你只需要一个API，比如：http://www.test.com/graphql即可。查询不同的内容仅需要改变post内容，不再需要维护多个api。（使用官方的demo进行演示：https://graphql.org/swapi-graphql）

比如查id为1的一个人的生日，可以这么查：

```
query{
  person(id:1,personID:1) {
    id
    birthYear
  }
}
```

```
{
  "data": {
    "person": {
      "id": "cGVvcGxlOjE=",
      "birthYear": "19BBY"
    }
  }
}
```

想查他的身高、发色可以这么查：

```
query{
  person(id:1,personID:1) {
    id
    birthYear
    height
    hairColor
  }
}
```

```
{
  "data": {
    "person": {
      "id": "cGVvcGxlOjE=",
      "birthYear": "19BBY",
      "height": 172,
      "hairColor": "blond"
    }
  }
}
```

我想查id为2的人的信息我可以这么查：

```
query{
  person(id:2,personID:2) {
    id
    birthYear
    height
    hairColor
  }
}
```

```
{
  "data": {
    "person": {
      "id": "cGVvcGxlOjI=",
      "birthYear": "112BBY",
      "height": 167,
      "hairColor": "n/a"
    }
  }
}
```

通过上面这个例子就可以看出graphql与REST API的区别，仅用一个API即可完成所有的查询操作。并且他的语法和结构都是以一个对象不同属性的粒度划分，简单好用。

基本属性

GraphQL的执行逻辑大致如下：

查询->解析->验证->执行

根据官方文档，主要的操作类型有三种：query（查询）、mutation（变更）、subscription（订阅），最常用的就是query，所有的查询都需要操作类型，除了简写查询语法。

类型语言TypeLanguage，type来定义对象的类型和字段，理解成一个数据结构，可以无关实现graphQL的语言类型。类型语言包括Scalar（标量）和Object（对象）两种。并且支持接口抽象类型。

Schema用于描述数据逻辑，Schema就是对象的合计，其中定义的大部分为普通对象类型。一定包括query，可能包含mutation，作为一个GraphQL的查询入口。

Resolver用于实现解析逻辑，当一个字段被执行时，相应的 resolver 被调用以产生下一个值。

内省查询

简单来说就是，GraphQL内置了接口文档，你可以通过内省的方法获得这些信息，如对象定义、接口参数等信息。

当使用者不知道某个GraphQL接口中的类型哪些是可用的，可以通过__schema字段来向GraphQL查询哪些类型是可用的。

```
{
  __schema {
    types {
      name
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Root"
        },
        {
          "name": "String"
        },
        {
          "name": "Int"
```

```
{
  __type(name: "Film") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```

具体可以参考GraphQL文档学习。

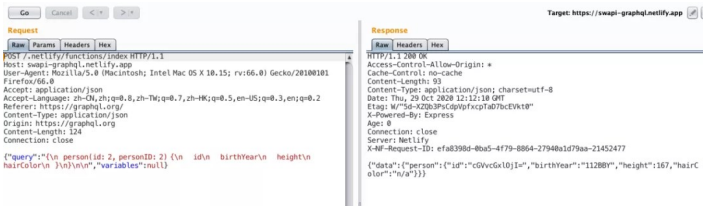
GraphQL中常见的问题

内省查询问题

这本来应该是仅允许内部访问，但配置错误导致任何攻击者可以获得这些信息。

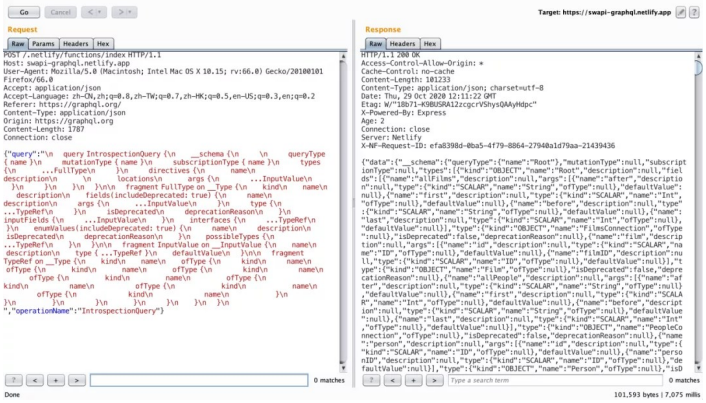
还是拿官网的demo来测试。

一个正常的查询请求如下。



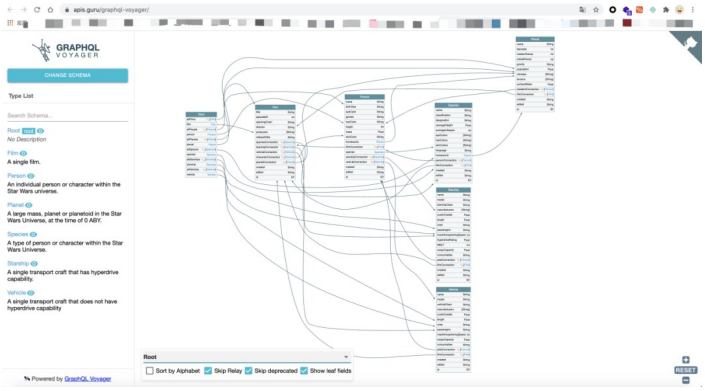
通过内省查询获得的数据如下：

```
{"query":"\n    query IntrospectionQuery {\r\n      __schema {\r\n
    queryType { name }\r\n        mutationType { name }\r\n        subscript
  ionType { name }\r\n        types {\r\n          ...FullType\r\n        }
  \r\n        directives {\r\n          name\r\n          description\r\n
          locations\r\n          args {\r\n            ...InputValue\r\n
        }\r\n        }\r\n      }\r\n    }\r\n\r\n    fragment FullType on
    __Type {\r\n      kind\r\n      name\r\n      description\r\n      field
  s(includeDeprecated: true) {\r\n        name\r\n        description\r\n
        args {\r\n          ...InputValue\r\n        }\r\n        type {\r\
  n          ...TypeRef\r\n        }\r\n        isDeprecated\r\n        dep
  recationReason\r\n      }\r\n      inputFields {\r\n        ...InputValue
  \r\n      }\r\n      interfaces {\r\n        ...TypeRef\r\n      }\r\n
     enumValues(includeDeprecated: true) {\r\n        name\r\n        descr
  iption\r\n        isDeprecated\r\n        deprecationReason\r\n      }\r\
  n      possibleTypes {\r\n        ...TypeRef\r\n      }\r\n    }\r\n\r\n
     fragment InputValue on __InputValue {\r\n      name\r\n      descripti
  on\r\n      type { ...TypeRef }\r\n      defaultValue\r\n    }\r\n\r\n
    fragment TypeRef on __Type {\r\n      kind\r\n      name\r\n      ofType
    {\r\n        kind\r\n        name\r\n        ofType {\r\n          kind\
  r\n          name\r\n          ofType {\r\n            kind\r\n
      name\r\n            ofType {\r\n              kind\r\n              nam
  e\r\n              ofType {\r\n                kind\r\n                na
  me\r\n                ofType {\r\n                  kind\r\n
       name\r\n                  ofType {\r\n                    kind\r\n
                  name\r\n                  }\r\n                }\r\n
```



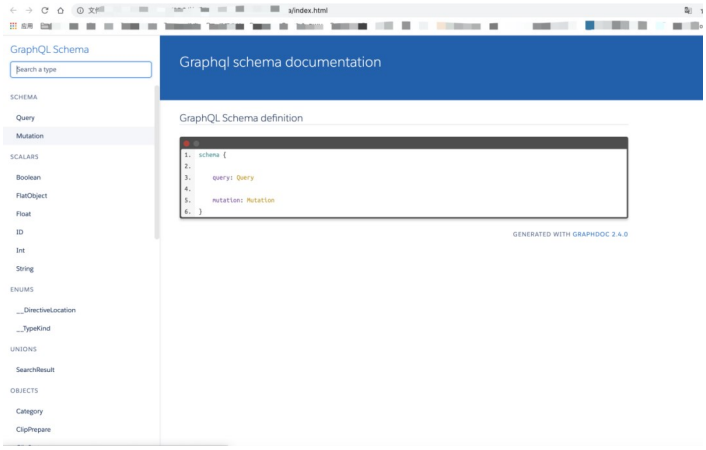
返回包返回的就是该API端点的所有信息。复制返回包到以下网址可以得到所有的对象定义、接口信息。

https://apis.guru/graphql-voyager/

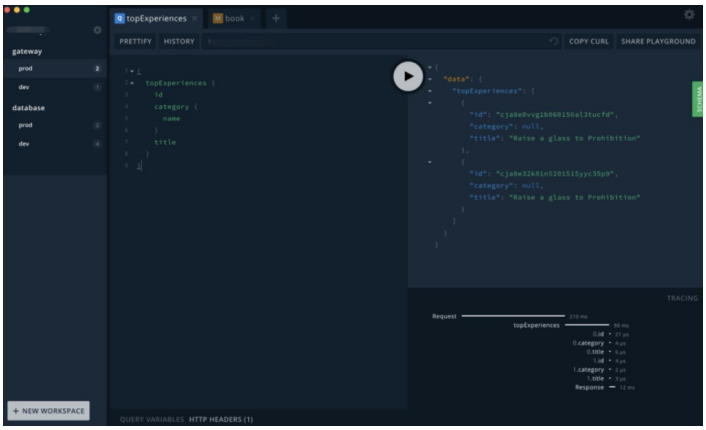


github也有很多工具可以直接绘制接口文档：

https://github.com/2fd/graphdoc



https://github.com/graphql/graphql-playground



这是garphql最常见的一类问题，通过这些文档我们就能很轻松的找到存在问题的对象了。通过遍历，即可发现很多安全问题。不过这个问题可以通过配置来解决，让攻击者无法获得敏感信息，或者其他攻击面。

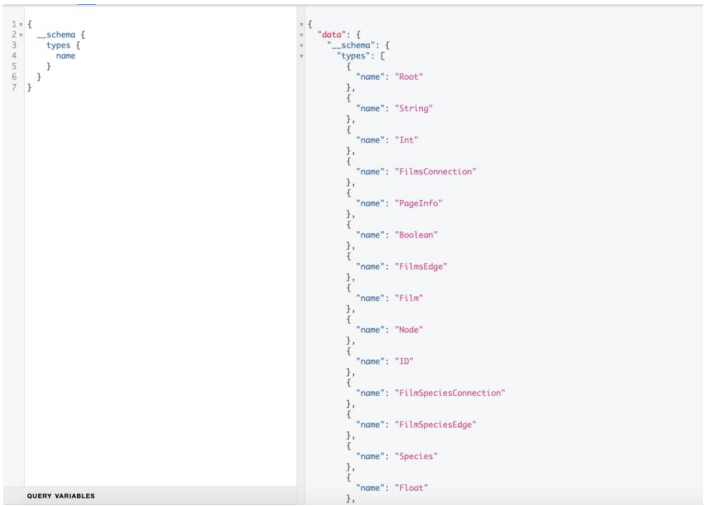
信息泄露

通过内省查询，我们可以得到很多后端接口的信息。有了这些信息通过排查便可能发现

更多的安全问题，比如信息泄露。

查询存在的类型：

```
{
  __schema {
    types {
      name
    }
  }
}
```



查询类型所有的字段：

```
{
  __type (name: "Query") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
          kind
        }
      }
    }
  }
}
```

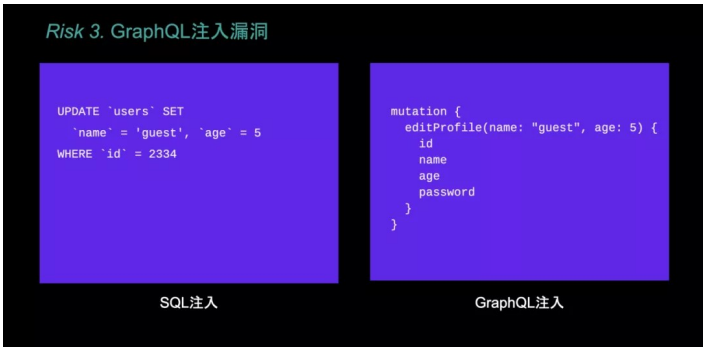
在查找字段里是否包含一些敏感字段：

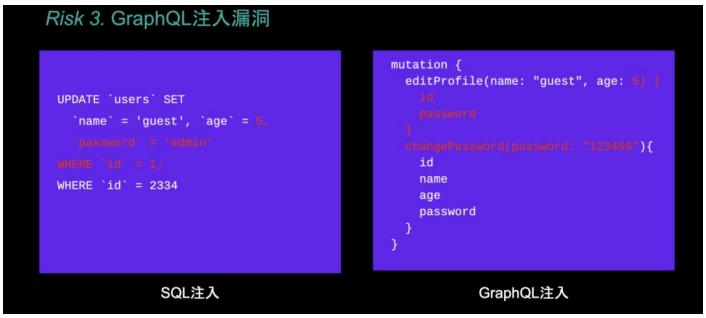
Email、token、password、authcode、license、key、session、secretKey、uid、address等。

除此以外还可以搜索类型中是否有edit、delete、remove、add等功能，来达到数据编辑、删除、添加的功能。

SQL注入

graphql的sql注入与一般的sql注入类似，都是可以通过构造恶意语句达到注入获取数据或改变查询逻辑的目的。p神在先知大会上讲过该类问题，借用p神的2张PPT。

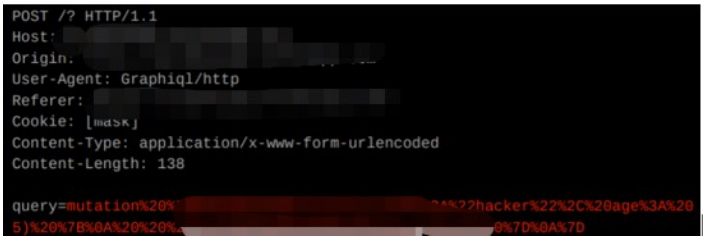




只有直接使用graphql进行查询才会出现的问题，正确的使用参数化查询，不会遇到sql注入的问题。

CSRF

在Express-GraphQL中存在CSRF漏洞。如果将Content-Type修改为 application/x-www-form-urlencoded ，再将POST请求包内容URL编码并生成csrf poc 即可实施csrf攻击，对敏感操作如mutation（变更）造成危害。



修复方式可以考虑将CORS配置为仅允许来自受信任域的白名单的请求，或者确保正在使用CSRF令牌.实施多种保护将降低成功攻击的风险.

嵌套查询拒绝服务

当业务的变量互相关联，如以下graphql定义为这样时，就可能无限展开，造成拒绝服务。

```
type Thread {
  messages(first: Int, after: String): [Message]
}

type Message {
  thread: Thread
}

type Query {
  thread(id: ID!): Thread
}
```

就有可能存在拒绝服务的风险。

```
query maliciousQuery {
  thread(id: "some-id") {
```

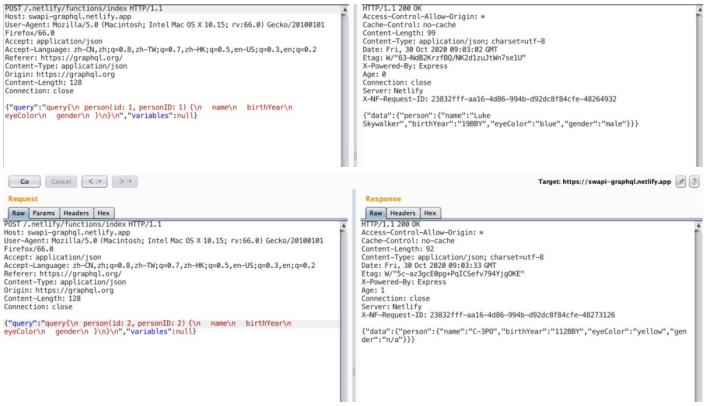
```
    messages(first: 99999) {
      thread {
        messages(first: 99999) {
          thread {
            messages(first: 99999) {
              thread {
                # ...repeat times 10000...
              }
            }
          }
        }
      }
    }
  }
}
```

就可能造成服务器拒绝服务。

修复方式可以考虑增加深度限制，使用graphql-depth-limit模块查询数量限制；或者使用graphql-input-number创建一个标量，设置最大为100

权限问题

graphql本身建议由业务层做权限控制，graphql作为一个单路由的API接口完成数据查询操作。开发者在使用时经常会忽略接口的鉴权问题。有时候客户端调用查询接口，直接传入了id等信息并未做好权限校验，就有可能存在水平越权。



修复方式建议在GraphQL和数据之间多加一个权限校验层，或者由业务自行实现权限校验。

总结

GraphQL技术由于其兼容restAPI，降低了API维护的成本已有很多企业在使用。可能存在的安全问题有：

1） 信息泄露

2） Sql注入

3） Csrf漏洞

4） 嵌套查询拒绝服务漏洞

5） 越权漏洞

6） 内省查询

在理解了GraphQL的工作原理和存在的问题后，大家工作或挖SRC过程中遇到这类技术可以有针对性的进行漏洞挖掘，本人也是第一次接触此类技术如有错误还请斧正。

未经允许不可引用

知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你

长按二维码关注 酒仙桥六号部队

PHP文件包含小总结

害、非法拘禁！

东北全面放开生育？官方回应