GraphQL 与 GraphQL 安全

图南
奇安信 A-TEAM

个人简介

- 图南
  - 奇安信 A-TEAM成员
  - 安全研究员
  - 曾经是一只程序猿（头发还在）
  - 邮箱：bibotai@qianxin.com

奇安信 A-TEAM

团队主要致力于Web渗透、APT攻防、对抗，前瞻性攻防工具预研。从底层原理、协议层面进行严肃、有深度的技术研究，深入还原攻与防的技术本质，曾多次率先披露 Windows域、Exchange、WebLogic等重大安全漏洞，第一时间发布相关漏洞安全风险通告及可行的处置措施并获得官方致谢。团队急需产品经理、安全开发和安全研究员。欢迎有意者加入！

01 | 说在前面的话

说在前面的话

- 共同作者：@gyyyy(猎户攻防实验室)、图南
- 后端： 
- 前端： 

02 | GraphQL简介

初识GraphQL

```
GET /products HTTP/1.1
Host: www.example.com
Accept: */*
User-Agent: example-user-agent
Connection: keep-alive
```

```
POST /orders HTTP/1.1
Host: www.example.com
Accept: */*
User-Agent: example-user-agent
Content-Type: application/json
Connection: keep-alive

{
  "data":"some data"
}
```

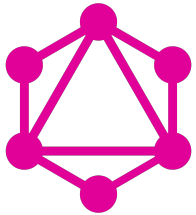
初识GraphQL

```
POST /graphql HTTP/1.1
Host: www.example.com
accept: */*
User-Agent: example-user-agent
content-type: application/json
Connection: keep-alive

{"operationName":null,"variables":{},"quer
y":"{\n  product(productId: \"1000\") {\n
_id\n    name\n  }\n}\n"}
```

```
POST /graphql HTTP/1.1
Host: www.example.com
accept: */*
User-Agent: example-user-agent
content-type: application/json
Connection: keep-alive

{"operationName":"CreateOrder","variable
s":{"data":"some data"},
"query":"mutation CreateOrder($data:
Data!) {\n  createOrder(data: $data)\n}\n"}
```

初识GraphQL

{ REST }



GraphQL

使用GraphQL的大客户们





GraphQL是什么

描述你的数据

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

请求你所要的数据

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

得到可预测的结果

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

GraphQL是什么

请求你所要的数据，不多不少

```
query {
  blog(blogId: "5b67230e2a755df312e6ce98") {
    _id
    type
  }
}
```

```
{
  "data": {
    "blog": [
      {
        "_id": "5b67230e2a755df312e6ce98",
        "type": "ARTICLE"
      }
    ]
  }
}
```

GraphQL是什么

获取多个资源，只用一个请求

```
query {
  blog(blogId: "5b67230e2a755df312e6ce98") {
    _id
    type
    title
    content
    desc
  }
}
```

```
{
  "data": {
    "blog": [
      {
        "_id": "5b67230e2a755df312e6ce98",
        "type": "ARTICLE",
        "title": "博客标题",
        "content": [
          "博客内容"
        ],
        "desc": "博客描述"
      }
    ]
  }
}
```

GraphQL是什么

描述所有的可能，类型系统

```
query {
  blog(blogId: "5b67230e2a755df312e6ce98") {
    _id
    type
    author{
      _id
      name
      email
      location
    }
    title
    content
    desc
  }
}
```

```
{
  "data": {
    "blog": [
      {
        "_id": "5b67230e2a755df312e6ce98",
        "type": "ARTICLE",
        "author": {
          "_id": "5cda7bfa979474558b9df07e",
          "name": "图南",
          "email": "tunan@example.com",
          "location": "Beijing,China"
        },
        "title": "博客标题",
        "content": [
          "博客内容"
        ],
        "desc": "博客描述"
      }
    ]
  }
}
```

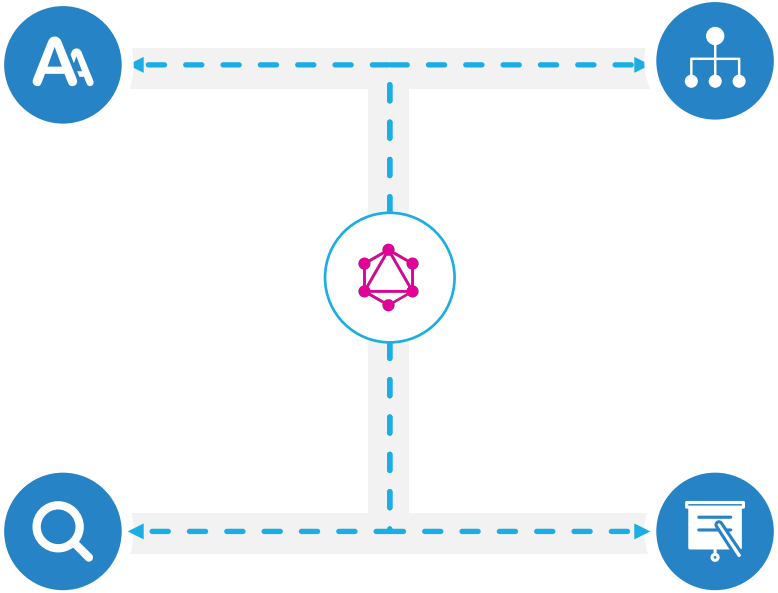
GraphQL核心组成部分

TYPE

用于描述接口的抽象数据模型，有**Scalar**（ 标量 ）和**Object**（ 对象 ）两种，**Object**由**Field**组成，同时**Field**也有自己的**Type**

SCHEMA

用于描述接口获取数据的逻辑，类比**RESTful**中的每个独立资源**URI**
一个 **GraphQL Schema** 中的最基本的组件是对象类型



QUERY

用于描述接口的查询类型，有**Query**（ 查询 ）、**Mutation**（ 更改 ）和**Subscription**（ 订阅 ）三种

RESOLVER

用于描述接口中每个**Query**的解析逻辑，部分**GraphQL**引擎还提供**Field**细粒度的**Resolver**

GraphQL核心组成部分

```
const typeDefs = gql `
  type Blog {
    title: String
    content: String
  }

  type Query {
    blogs: [Blog]
  }
`;
```

Type定义描述要查询数据的“形状”，并指定GraphQL服务器获取数据的方式，Schema包含在Type中， Schema的最基本的组件是对象类型（Object type）

Query是所有GraphQL查询的根

```
const resolvers = {
  Query: {
    blogs: () => blogs,
  },
};
const server = new ApolloServer({
  typeDefs,
  resolvers
});
```

Resolver定义用怎样的技术方式从Schema中取出对应类型的数据

GraphQL VS. RESTful VS. 参数型接口

参数型接口和RESTful接口

请求	响应
GET /blog?id={blogId}	{"data": {"... blog data ..."}}
GET /author?id={authorId}	{"data": {"... author data ..."}}
GET /blog/{blogId}	{"data": {"... blog data ..."}}
GET /author/{authorId}	{"data": {"... author data ..."}}

GraphQL VS. RESTful VS. 参数型接口

GraphQL接口

请求	响应

POST /graphql
 	
{"query": "{\n blog(blogId: \"{ blogId}\") {\n author {\n field\n }\n field \n }\n}\n"}
	{
	"data": {
	"blog": [{
	"author": {
	"field": "..."
	}
	"field": "..."
	}]
	}
	}

GraphQL VS. RESTful

RESTful

GraphQL VS. RESTful

GraphQL

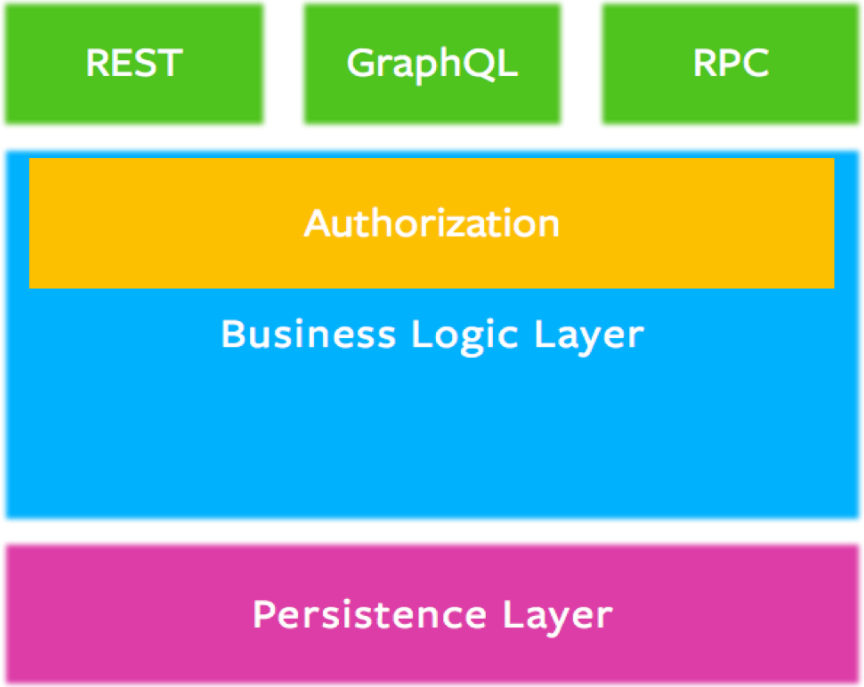
03 | GraphQL安全问题

GraphQL安全漏洞

275	Account takeover via leaked session cookie	By haxta4ok00 to HackerOne	Resolved · High · $20,000.00	disclosed 2 days ago
98	Disclosure of `payment_transactions` for programs via GraphQL query	By msdian7 to HackerOne	Resolved · Medium · $2,500.00	disclosed 4 days ago
123	Private program disclosure via `vpn_suspended` GraphQL query	By unknown_person to HackerOne	Resolved · None · $2,500.00	disclosed about 1 month ago
46	Unauthenticated read and write access to ALL endpoints of a store is possible for removed staff members who had "Apps" permission	By mariogh to Shopify	Resolved · Medium · $1,500.00	disclosed 2 months ago
882	Confidential data of users and limited metadata of programs and reports accessible via GraphQL	By yashrs to HackerOne	Resolved · Critical · $20,000.00	disclosed 10 months ago
8	H1514 Get access to non public information by pivoting with graphql queries	By emitrani to Shopify	Resolved · Low · $1,500.00	disclosed about 1 month ago
69	Private information exposed through GraphQL filters	By reigertje to HackerOne	Resolved · Medium	disclosed 5 months ago
130	Bug in GraphQL and API integration leads to limited user address disclosure	By loxiran to Starbucks	Resolved · High	disclosed 9 months ago

身份认证与权限控制不当

GraphQL和其他API技术一样，需要身份认证和鉴权



身份认证与权限控制不当

Rojan Rijal (rijalrojan)

6613	79th	6.58	95th	22.94	94th
Reputation	Rank	Signal	Percentile	Impact	Percentile

42

#397130 **Unauthenticated access to Zendesk tickets through athena-flex-production.shopifycloud.com Okta bypass**

Share:

State	Resolved (Closed)	Severity	Critical (9.8)
Disclosed	September 20, 2018 12:15am +0800	Participants	
Reported To	Shopify	Visibility	Disclosed (Full)
Asset	partners.shopify.com (Domain)		
Weakness	Improper Authentication - Generic		
Bounty	$5,000		

Collapse

SUMMARY BY SHOPIFY

@rijalrojan discovered an application and endpoint under `athena-flex-production.shopifycloud.com` that exposed metadata and contents of our Zendesk tickets. Within a couple hours, we had put it behind an OAuth portal to mitigate the issue. After an internal investigation revealed no evidence of malicious access to the data, we rewarded him with the highest bounty available under our non-core authentication bypass category ($5,000). We did this despite the app being out of scope because of the severity of the information disclosure. As always, out-of-scope rewards are at our sole discretion.

身份认证与权限控制不当

TIMELINE

rijalrojan submitted a report to **Shopify**. Aug 20th (about 1 year ago)

Summary

athena-flex-production.shopifycloud.com seems to be an internal system that Shopify uses because it redirects user to Okta login. During this however, I noticed that it first returns 200 and then does a redirect meaning some part of the website loads before redirecting. With this, I was able to get the JS being used in the system. Through the JS file, I found a path that allows GraphQL queries thus resulting in a full dump of Zendesk ticket information.

- 在一个JavaScript文件中暴露了一个GraphQL接口路径

When you originally go to athena-flex-production.shopifycloud.com you will find that it will redirect to Okta. However if you do `view-source:athena-flex-production.shopifycloud.com` in Chrome, it will show that the website loads momentarily. In one of the script src, there is this link requested by the website:

- 通过这个GraphQL接口路径在未经身份认证的前提下查询到了很多关键数据

When you beautify the JS file you will notice it has some query data that can be used at the /graphql endpoint. After I got this, I started to play around with the GraphQL schema and see what I could gain access to.

```
{"query": "query getRecentTicketsQuery($domain: String) {\n    shop(myshopifyDomain: $domain) {\n      zendesk {\n
```

What this query says is: Return last 5 tickets with description, reporter name and subject of the ticket that contain domain ok.myshopify.com. Once the query was done, it responded with 9,259 bytes of JSON response that contained extremely critical data.

I don't want to paste the data here for obvious reason but I am attacking the file here so you can delete it by contact support@hackerone.com later if you wish to disclose the report.

身份认证与权限控制不当

- 一个获取用户地址簿的GraphQL查询语句可泄露部分用户地址簿信息

身份认证与权限控制不当

- **组内普通成员可通过此漏洞越权成为主持人**

身份认证与权限控制不当

GraphQL身份认证无效和权限控制不当问题频发的原因

01

GraphQL多了一个中间层对它定义的查询语言进行语法解析执行等操作

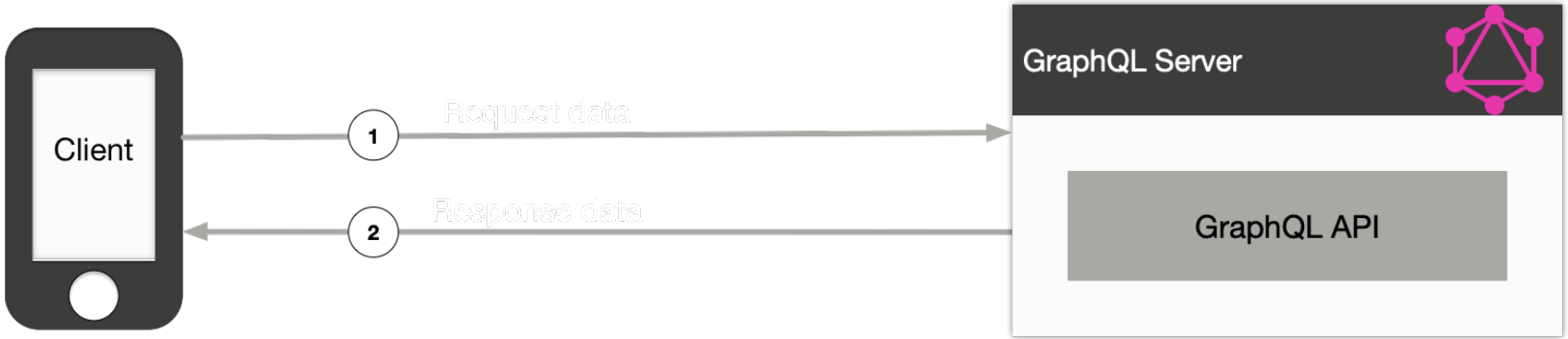
02

Schema、Resolver等种种定义会让开发者对它的存在感知较大，间接的增加了对它理解的复杂度

03

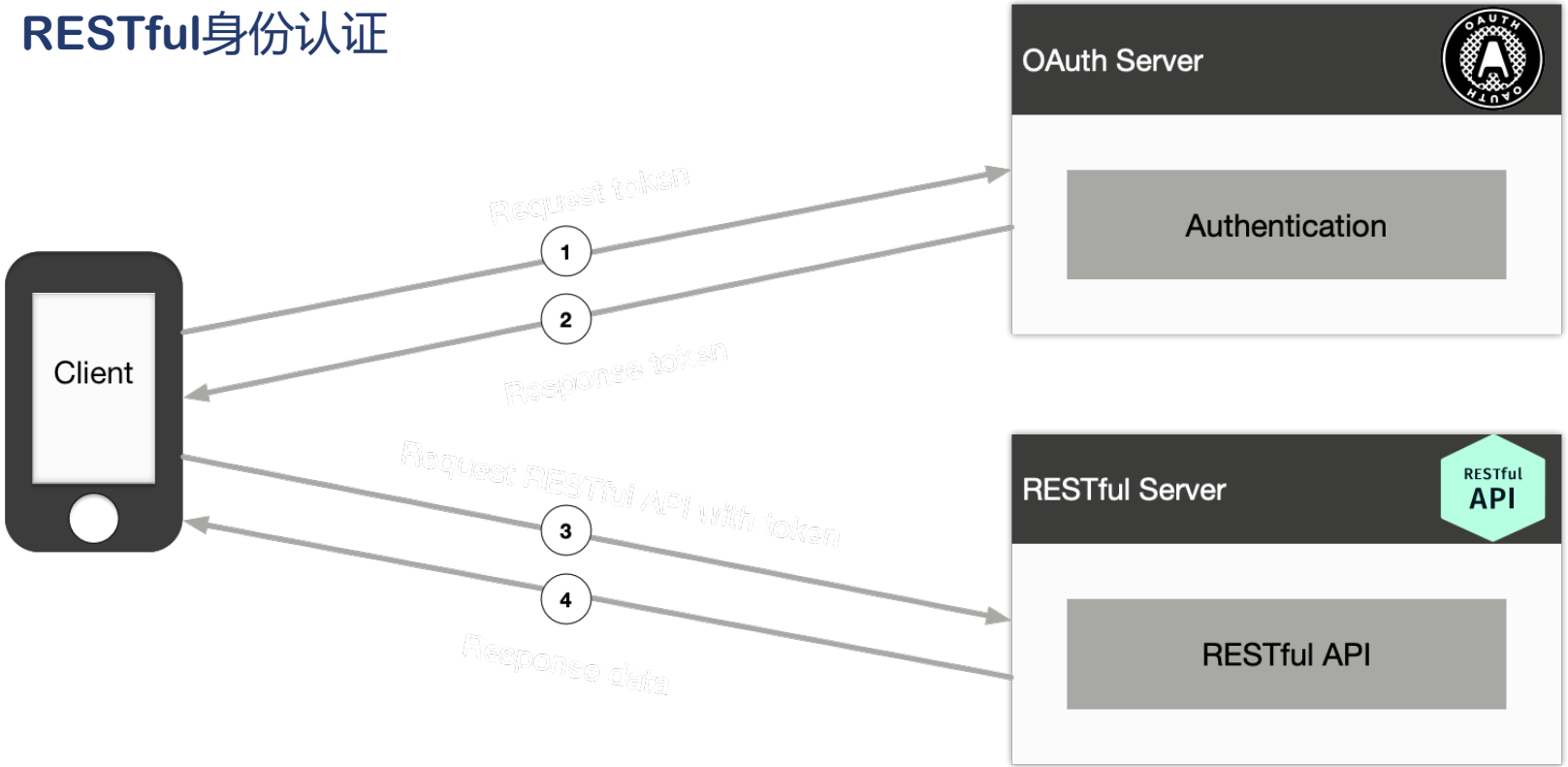
单路由形态，通过不同的field查询得到不同的结果，开发者需要对每个field的resolve做权限控制

身份认证无效


Client
1
Request data
2
Response data
GraphQL Server
GraphQL API

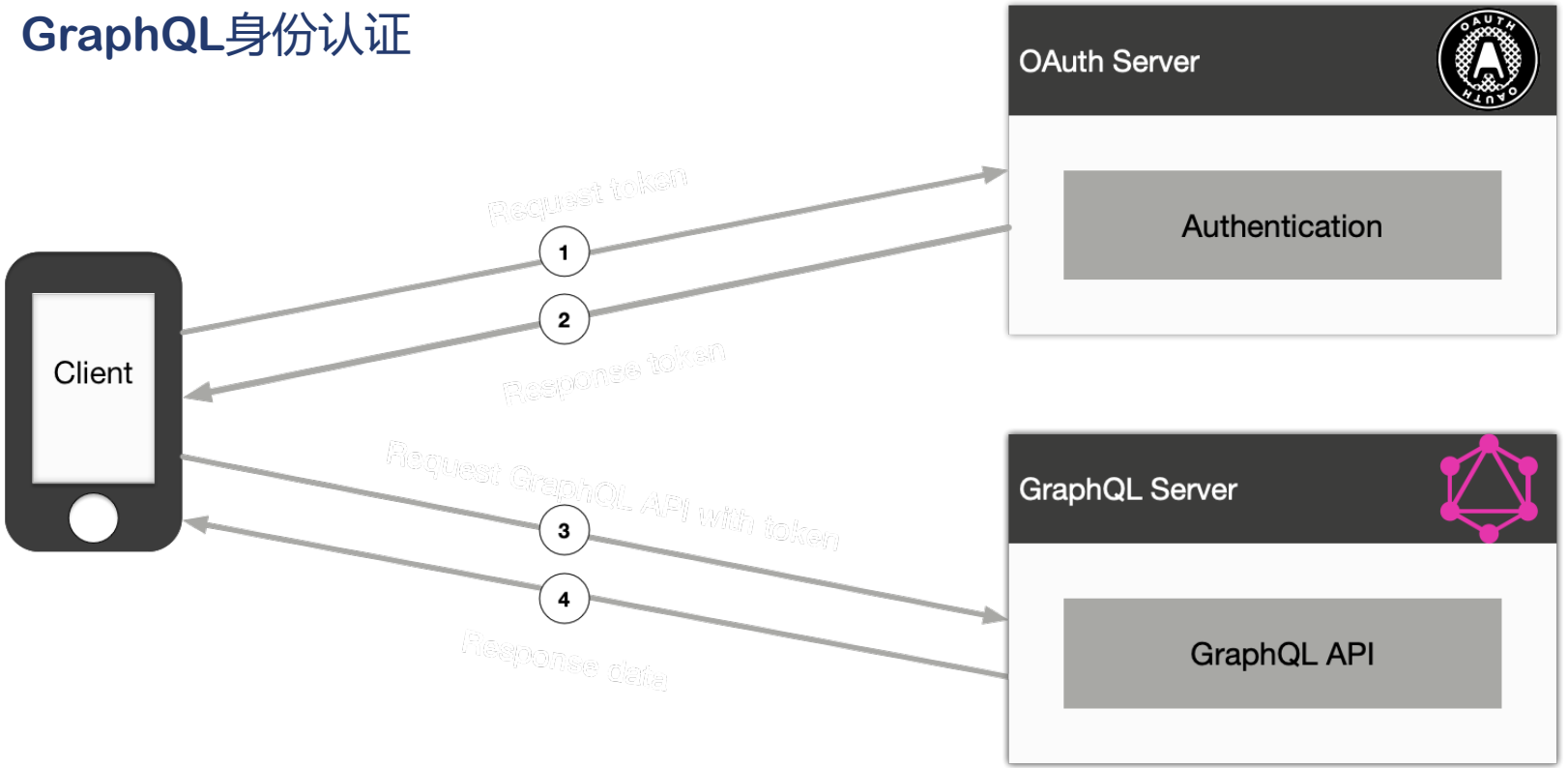
GraphQL认证无效解决方案

RESTful身份认证



GraphQL认证无效解决方案

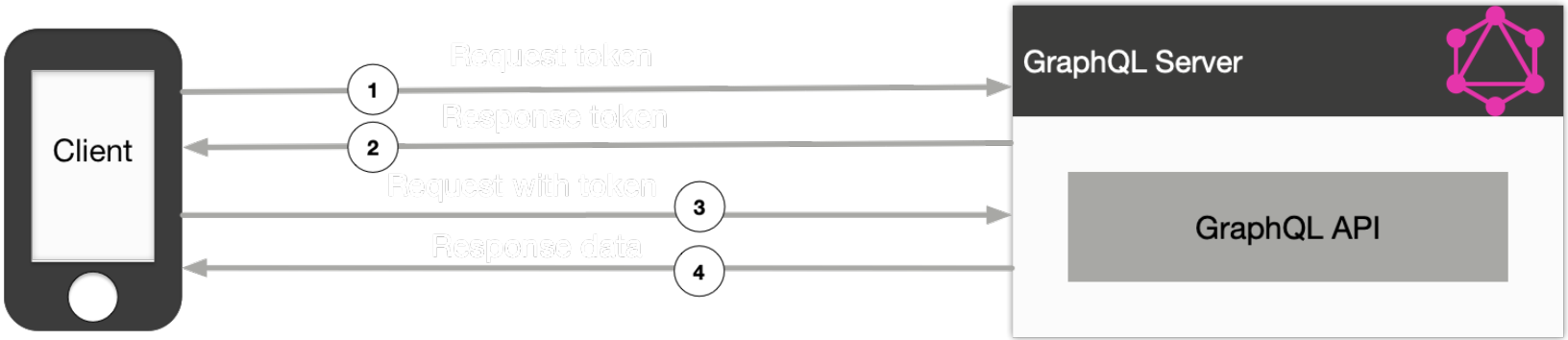
GraphQL身份认证



GraphQL认证无效解决方案

```
//...omit...
router.post('/login', LoginController.login);
router.post('/register', LoginController.register);

app.use(koajwt({
  secret: 'your secret'
}).unless({
  path: [/^\/public/, '/login', '/register']
}));

const server = new ApolloServer({
  typeDefs: schemaText,
  resolvers: resolverMap,
  context: ({
    ctx
  }) => ({
    ...ctx,
    ...app.context
  })
});

server.applyMiddleware({
  app
});

app.listen({
  port: 4000
}, () => console.log(`🚀  Server ready at http://localhost:4000${server.graphqlPath}`), );
//...omit...
```

GraphQL认证无效解决方案



GraphQL认证无效解决方案

```
import bcrypt from 'bcryptjs'
import jsonwebtoken from 'jsonwebtoken'

export const login = async (_, args, context) => {
  console.log('login...');
  const db = await context.getDb()
  const {
    username,
    password
  } = args
  const user = await db.collection('User').findOne({
    username: username
  });
  if (await bcrypt.compare(password, user.password)) {
    return {
      message: '登录成功',
      token: jsonwebtoken.sign({
        user: user,
        exp: Math.floor(Date.now() / 1000) + (60 * 60 * 12),
      }, 'your secret'),
    }
  }
}
```

```
type Query {
  login(
    username: String!
    password:String!
  ): LoginMsg
}

type LoginMsg{
  message:String
  token:String
}
```

GraphQL认证无效解决方案

```
const server = new ApolloServer({
  typeDefs: schemaText,
  resolvers: resolverMap,
  context: ({ ctx }) => {
      const token = ctx.req.headers.authorization || '';
      const user = getUser(token);
      return {
          ...user,
          ...ctx,
          ...app.context
      };
  },
});
```

并发症一：内省导致的信息泄露

GraphQL自带强大的内省机制

__schema可查询所有可用对象

__type可查询指定对象的所有字段

并发症一：内省导致的信息泄露

__schema的用法

```
query {
  __schema {
    queryType {
      name
      fields {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query",
        "fields": [
          {
            "name": "_dummy"
          },
          {
            "name": "author"
          },
          {
            "name": "blog"
          },
          {
            "name": "userInfo"
          },
```

并发症一：内省导致的信息泄露

__type的用法

```
query {
  __type(name: "Blog") {
    name
    fields {
      name
      type {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "__type": {
      "name": "Blog",
      "fields": [
        {
          "name": "_id",
          "type": {
            "name": null
          }
        },
        {
          "name": "type",
          "type": {
            "name": "BlogType"
          }
        },
        {
          "name": "avatar",
          "type": {
            "name": "String"
          }
```

并发症二： 非预期的字段

同一个接口给不同的权限职能属性的角色使用

•正常情况下每一个角色只查询接口的部分字段

•但如果不对字段权限进行控制，即可被恶意利用查询出全部字段

并发症二： 非预期的字段

容易查询出非预期字段的原因

01

GraphQL是根据前端请求的字段进行数据回传

02

后端Resolver的响应包含对应字段即可

03

后端字段扩展对前端无感知无影响

并发症二： 非预期的字段

举个  假设有如下需求：

需求1：管理员可查看用户的相关信息（需要用户名、邮箱、创建日期等字段）

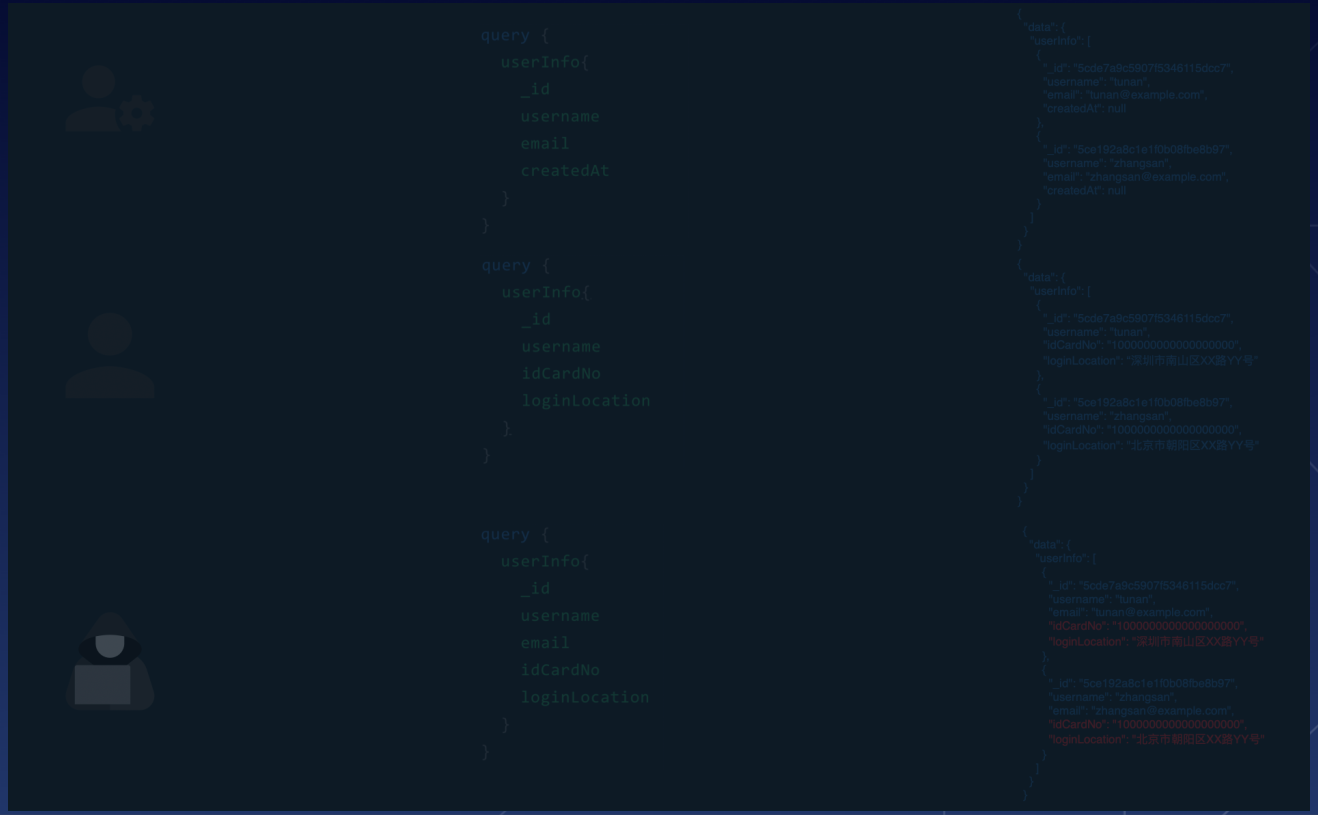
需求2：运营人员需要统计用户数据，输出报表（需要用户名、身份证号、登录地点等字段）

并发症二： 非预期的字段

设计Schema如下：

```
# 用户信息
type UserInfo{
  _id: String!
  # 用户名
  username:String
  # 邮箱
  email:String
  # 身份证号
  idCardNo:String
  # 登录地点
  loginLocation:String
  # 创建时间
  createdAt:Date
  # 更新时间
  updatedAt:Date
}
```

并发症二：非预期字段


query {
userInfo{
_id
username
email
createdAt
}
}
{
"data": {
"userInfo": [
{
"_id": "5cde7a9c5907f5346115dcc7",
"username": "lunan",
"email": "lunan@example.com",
"createdAt": null
},
{
"_id": "5ce192a8c1e1f0b08fbe8b97",
"username": "zhangsan",
"email": "zhangsan@example.com",
"createdAt": null
}
]
}
}
query {
userInfo{
_id
username
idCardNo
loginLocation
}
}
{
"data": {
"userInfo": [
{
"_id": "5cde7a9c5907f5346115dcc7",
"username": "lunan",
"idCardNo": "100000000000000000",
"loginLocation": "深圳市南山区XX路YY号"
},
{
"_id": "5ce192a8c1e1f0b08fbe8b97",
"username": "zhangsan",
"idCardNo": "100000000000000000",
"loginLocation": "北京市朝阳区XX路YY号"
}
]
}
}
query {
userInfo{
_id
username
email
idCardNo
loginLocation
}
}
{
"data": {
"userInfo": [
{
"_id": "5cde7a9c5907f5346115dcc7",
"username": "lunan",
"email": "lunan@example.com",
"idCardNo": "100000000000000000",
"loginLocation": "深圳市南山区XX路YY号"
},
{
"_id": "5ce192a8c1e1f0b08fbe8b97",
"username": "zhangsan",
"email": "zhangsan@example.com",
"idCardNo": "100000000000000000",
"loginLocation": "北京市朝阳区XX路YY号"
}
]
}
}

并发症三：『废弃』的字段

GraphQL为字段删除提供了『废弃』方案

如果一个字段不再使用，可以标识字段『废弃』

•但如果不对『废弃』字段的Resolver进行修改，依然可以查询到废弃字段的内容

并发症三：『废弃』的字段

刚才的  继续：开发者意识到了问题，决定废弃敏感字段

```
# 用户信息
type UserInfo{
  _id: String!
  # 用户名
  username:String
  # 邮箱
  email:String
  # 身份证号
  idCardNo:String @deprecated(reason:"安全问题")
  # 登录地点
  loginLocation:String @deprecated(reason:"安全问题")
  # 创建时间
  createdAt:Date
  # 更新时间
  updatedAt:Date
}
```

并发症三：『废弃』的字段

使用__type内省仍然可以将废弃

```
query {
  __type(name: "UserInfo") {
    name
    fields {
      name
      type {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "__type": {
      "name": "UserInfo",
      "fields": [
        {
          "name": "_id",
          "type": {
            "name": null
          }
        },
        {
          "name": "username",
          "type": {
            "name": "String"
          }
        },
        {
          "name": "email",
          "type": {
            "name": "String"
          }
```

并发症三：『废弃』的字段

由于开发者没有对Resolver做修改，废弃字段仍然可以正常参与查询

```
query{
  userInfo{
    _id
    username
    email
  }
}
```

```
{
  "data": {
    "userInfo": [
      {
        "_id": "5cde7a9c5907f5346115dcc7",
        "username": "tunan",
        "email": "tunan@example.com"
      },
      {
        "_id": "5ce192a8c1e1f0b08fbe8b97",
        "username": "zhangsan",
        "email": "zhangsan@example.com"
      }
    ]
  }
}
```

GraphQL注入

有语法就会有解析，有解析就会有结构和顺序，有结构和顺序就会有注入 —— @gyyyy

GraphQL注入

再举个  前端使用变量构建带参查询语句，通过用户名查询用户

```
const name = props.match.params.name;
const queryUser = gql `{
    user(username: ${name}) {
        _id
        username
        email
    }
}`
```

")%7Busername%7Dhack%3Auser(username%3A"admin")%7Bpassword%23

GraphQL注入

GraphQL语句的结构被改变

```
{
  user(username: "") {
    username
  }
  hack: user(username: "admin") {
    password#) {
    _id
    username
    email
  }
}
```

GraphQL注入解决方案

参数值以变量的形式传入，由解析器实时赋值解析

```
const name = props.match.params.name;
const queryUser = gql `{
  query User ($name: String!)
    user(username: $name) {
      _id
      username
      email
    }
}`

client.query({
  query: queryUser,
  variable: {
    name
  }
})
```

拒绝服务

GraphQL允许对象间的嵌套关系存在

•如果不对嵌套深度进行限制，就会被攻击者利用进行拒绝服务攻击

拒绝服务

举最后一个 

需求1：查询所有文章，返回内容中包含作者信息

需求2：查询作者信息，返回内容中包含此作者写的所有文章

拒绝服务

定义Blog和Author并构建Query

```
type Blog {
    _id: String!
    type: BlogType
    avatar: String
    title: String
    content: [String]
    author: Author
    # ...omit...
}

type Author {
    _id: String!
    name: String
    blog: [Blog]
    # ...omit...
}
```

```
extend type Query {
    blogs(
        blogId: ID
        systemType: String!
    ): [Blog]
}

extend type Query {
    author(
        _id: String!
    ): Author
}
```

拒绝服务

可构造如下恶意查询如下

```
query GetBlogs($blogId: ID, $systemType: String!) {
    blogs(blogId: $blogId, systemType: $systemType) {
        _id
        title
        type
        content
        author {
            name
            blog {
                author {
                    name
                    blog {
                        author {
                            name
                            blog {
                                author {
                                    name
                                    # ...omit...
                                }
                            }
                        }
                    }
                }
                title
                createdAt
                publishedAt
            }
        }
        publishedAt
    }
}
```

拒绝服务解决方案

限制查询深度，同时在设计GraphQL接口时应尽量避免出现此类问题

```
// ...omit...
import depthLimit from 'graphql-depth-limit';
// ...omit...
const server = new ApolloServer({
  typeDefs: schemaText,
  resolvers: resolverMap,
  context: ({
    ctx
  }) => {
    const token = ctx.req.headers.authorization || '';
    const user = getUser(token);
    console.log('user', user)
    return {
      ...user,
      ...ctx,
      ...app.context
    };
  },
  validationRules: [depthLimit(10)]
});
// ...omit...
```

04 结语

总结

- GraphQL是什么
- GraphQL与RESTful的区别
- 安全问题1：身份认证与权限控制不当
- 安全问题2：GraphQL注入
- 安全问题3：拒绝服务

他只是个接口

- XSS
- SQL注入
- NoSQL注入
- CSRF
- 远程命令执行
- ……

```
query GetAllUsers($filter: String!) {
    users(filter: $filter) {
        _id
        username
        email
    }
}
```

{"filter": "' or ''='"}

{"filter": "{\"$ne\": null}"}

我只是抛了个砖

- 关于GraphQL，我们还可以研究：
- 不同语言的实现
- 引擎内部执行流程
- 语法解析过程
- 校验和数据编解码
- ……

Geekbang. 极客邦科技 | TGO 鲲鹏會

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

900+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

Geekbang. 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

THANKS

GMTC
全球大前端技术大会

欢迎关注我们的微信公众号